Making Random Choices Invisible to the Scheduler

Konstantinos Chatzikokolakis, Catuscia Palamidessi

Making Random Choices Invisible to the Scheduler[☆]

Konstantinos Chatzikokolakis[a], Catuscia Palamidessi[b]

[a] *Eindhoven University of Technology, The Netherlands*
[b] *INRIA and LIX, École Polytechnique, Palaiseau, France*

Abstract

When dealing with process calculi and automata which express both nondeterministic and probabilistic behavior, it is customary to introduce the notion of scheduler to resolve the nondeterminism. It has been observed that for certain applications, notably those in security, the scheduler needs to be restricted so not to reveal the outcome of the protocol's random choices, or otherwise the model of adversary would be too strong even for "obviously correct" protocols. We propose a process-algebraic framework in which the control on the scheduler can be specified in syntactic terms, and we show how to apply it to solve the problem mentioned above. We also consider the definition of (probabilistic) may and must preorders, and we show that they are precongruences with respect to the restricted schedulers. Furthermore, we show that all the operators of the language, except replication, distribute over probabilistic summation, which is a useful property for verification.

1. Introduction

Security protocols, in particular those for anonymity and fair exchange, often use randomization to achieve their goals. Since they usually involve more than one agent, they also give rise to concurrent and interactive activities that can be best modeled by nondeterminism. Thus it is convenient to specify them using a formalism which is able to represent both *probabilistic* and *nondeterministic* behavior. Formalisms of this kind have been explored in both Automata Theory ([1, 2, 3, 4, 5]) and in Process Algebra ([6, 7, 8, 9, 10, 11]). See also [12, 13] for comparative and more inclusive overviews.

Due to the presence of nondeterminism, in such formalisms it is not possible to define the probability of events in *absolute* terms. We need first to decide how each nondeterministic choice during the execution will be resolved. This decision function is called *scheduler*. Once the scheduler is fixed, the behavior

[☆]This work has been partially supported by the ANR-09-BLAN-0169-01 project PANDA, the INRIA DREI Équipe Associée PRINTEMPS, and the STW/Sentinels project PEARL.

Email addresses: k.chatzikokolakis@tue.nl (Konstantinos Chatzikokolakis), catuscia@lix.polytechnique.fr (Catuscia Palamidessi)

of the system (*relatively* to the given scheduler) becomes fully probabilistic and a probability measure can be defined following standard techniques.

It has been observed by several researchers that in security the notion of scheduler needs to be restricted, or otherwise any secret choice of the protocol could be revealed by making the choice of the scheduler dependent on it. This issue was for instance one of the main topics of discussion at the panel of CSFW 2006. We illustrate it here with an example on anonymity. We use the standard CCS notation, plus a construct of probabilistic choice $P +_p Q$ representing a process that evolves into P with probability p and into Q with probability $1-p$.

The system *Sys* consists of a receiver R and two senders S, T communicating via private channels a, b respectively. Which of the two senders is successful is decided probabilistically by R. After reception, R sends a signal *ok*.

$$\begin{aligned} R &\triangleq a.\overline{ok}.0 \ +_{0.5} \ b.\overline{ok}.0 \\ S &\triangleq \bar{a}.0 \\ T &\triangleq \bar{b}.0 \\ Sys &\triangleq (\nu a)(\nu b)(R \mid S \mid T) \end{aligned}$$

The signal *ok* is public, but since it is the same in both cases, in principle an external observer should not be able to infer from it the identity of the sender (S or T). So the system should be anonymous. However, consider a team of two attackers A and B defined as

$$A \triangleq ok.\bar{s}.0 \qquad\qquad B \triangleq ok.\bar{t}.0$$

and consider the parallel composition $Sys \mid A \mid B$. We have that, under certain schedulers, the system is no longer anonymous. More precisely, a scheduler could leak the identity of the sender via the channels s, t by forcing R to synchronize with A on *ok* if R has chosen the first alternative, and with B otherwise. In this case, an output on the public channel s (resp. t) reveals that S (resp. T) was the sender. This is possible because in general a scheduler can see the whole history of the computation, in particular the random choices, even those which are supposed to be private. Note that the visibility of the synchronization channels to the scheduler is not crucial for this example: we would have the same problem, for instance, if S, T were both defined as $\bar{a}.0$, R as $a.\overline{ok}.0$, and *Sys* as $(\nu a)((S \ +_{0.5} \ T) \mid R)$.

The above example demonstrates that, with the standard definition of scheduler, it is not possible to represent a truly private random choice (or a truly private nondeterministic choice, for the matter) with the current probabilistic process calculi. This is a clear shortcoming when we want to use these formalisms for the specification and verification of security protocols.

There is another issue related to verification: a private choice has certain algebraic properties that would be useful in proving equivalences between processes. In fact, if the outcome of a choice remains private, then it should not matter at which point of the execution the process makes such choice, until it actually uses it. Consider for instance A and B defined as follows

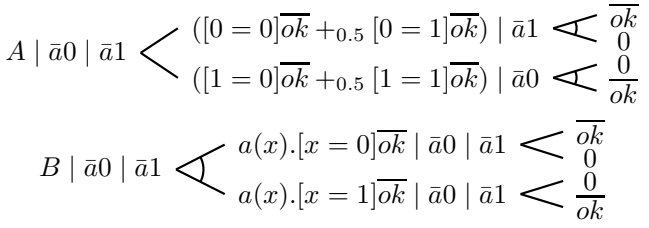

Figure 1: Execution trees for $A \mid C$ and $B \mid C$

$$A \triangleq a(x).([x=0]\overline{ok} \;+_{0.5}\; [x=1]\overline{ok}) \qquad B \triangleq a(x).[x=0]\overline{ok} \;+_{0.5}\; a(x).[x=1]\overline{ok}$$

Process A receives a value and then decides randomly whether it will accept the value 0 or 1. Process B does exactly the same thing except that the choice is performed before the reception of the value. If the random choices in A and B are private, intuitively we should have that A and B are equivalent ($A \approx B$). This is because it should not matter whether the choice is done before or after receiving a message, as long as the outcome of the choice is completely invisible to any other process or observer. However, consider the parallel context $C = \overline{a}0 \mid \overline{a}1$. Under any scheduler, A has probability at most $1/2$ to perform $\overline{ok}$. With B, on the other hand, the scheduler can choose between $\overline{a}0$ and $\overline{a}1$ based on the outcome of the probabilistic choice, thus making the maximum probability of $\overline{ok}$ equal to 1. The execution trees of $A \mid C$ and $B \mid C$ are shown in Figure 1.

In general when $+_p$ represents a private choice we would like to have

$$C[P +_p Q] \approx C[\tau.P] +_p C[\tau.Q] \tag{1}$$

for all processes P, Q and all contexts C *not containing replication (or recursion).* In the case of replication the above cannot hold since $!(P +_p Q)$ makes available each time the choice between P and Q, while $(!\tau.P) +_p (!\tau.Q)$ chooses once and for all which of the two (P or Q) should be replicated. Similarly for recursion. The reason why we need a τ is explained in Section 5.

The algebraic property (1) expresses in an abstract way the privacy of the probabilistic choice. Moreover, this property is also useful for the verification of security properties. The interested reader can find in [14] an application to a fair exchange protocol, as an example. In principle (1) should be useful for any kind of verification in the process algebra style.

We propose a process-algebraic approach to the problem of hiding the outcome of random choices. Our framework is based on a calculus obtained by adding to CCS an internal probabilistic choice construct[1]. This calculus, to

[1]We actually consider a variant of CCS where infinite behavior is expressed by replicated

which we refer as CCS_p, is a variant of the one studied in [11], the main differences being that we use replicated input instead of recursion, and we lift some restrictions that were imposed in [11] to obtain a complete axiomatization. The semantics of CCS_p is given in terms of Segala's *simple probabilistic automata* ([4, 7]).

In order to limit the power of the scheduler, we extend CCS_p with terms representing explicitly the notion of scheduler. The latter interact with the original processes via a labeling system. This will allow to specify at the syntactic level (by a suitable labeling) which choices should be visible to schedulers, and which ones should not.

1.1. Contribution

The main contributions of this paper are:

- A process calculus CCS_σ in which the scheduler is represented as a process, and whose power can therefore be controlled at the syntactic level.
- The adaptation of the standard notions of probabilistic testing preorders to CCS_σ and the "sanity check" that they are still precongruences. For must testing, we additionally require that the occurrences of $+$ in the context are guarded, otherwise we have the problem that P and $\tau.P$ are must equivalent, but $Q + P$ and $Q + \tau.P$ are not. This is typical for the plus operator of CCS: usually it does not preserve weak equivalences.
- The proof that, under suitable conditions on the labelings of C, $\tau.P$ and $\tau.Q$, CCS_σ satisfies the property expressed by (1), where $\approx$ is probabilistic testing equivalence.
- An application of CCS_σ to an extended anonymity example (the Dining Cryptographers Protocol, DCP). We also briefly outline how to extend CCS_σ so to allow the definition of private nondeterministic choice, and we apply it to the DCP with a nondeterministic master. To our knowledge this is (together with [16]) the first formal treatment of the scheduling problem in DCP and the first formalization of a nondeterministic master for the (probabilistic) DCP.

1.2. Related work

The works that are most closely related to ours are [17, 18, 16]. The authors of [17, 18] consider probabilistic automata and introduce a restriction on the scheduler to the purpose of making them suitable to applications in security protocols. Their approach is based on dividing the actions of each component

input-prefixed processes rather than by recursion, since this choice simplifies the formalization of schedulers. This version of CCS is not equivalent to the original one because replication corresponds to recursion with static scope while recursion in CCS has dynamic scope ([15]), however the scoping issues are orthogonal to those investigated in this paper.

of the system in equivalence classes (*tasks*). The order of execution of different tasks is decided in advance by a so-called *task scheduler*. The remaining nondeterminism within a task is resolved by a second scheduler, which models the standard *adversarial scheduler* of the cryptographic community. This second entity has limited knowledge about the other components: it sees only the information that they communicate during execution.

In [16] the authors define a notion of admissible scheduler by introducing an equivalence relation on the nodes of the execution tree, and requiring that an admissible scheduler maps two equivalent nodes into bisimilar steps. Both our paper and [16] have developed, independently, the solution to the problem of the scheduler in the Dining Cryptographers as an example of application to security.

Another work along these lines is [19], which uses partitions on the state-space to obtain partial-information schedulers. However [19] considers a synchronous parallel composition, so the setting is rather different from [17, 18, 16] and ours.

Our approach is in a sense *dual* to the above ones. Instead of defining a restriction on the class of schedulers, we provide a way to specify that a choice is transparent to the scheduler. We achieve this by introducing labels in process terms, used to represent both the nodes of the execution tree and the next action or step to be scheduled. We make two nodes indistinguishable to schedulers, and hence the choice between them private, by associating to them the same label. Furthermore, in contrast with [17, 18], our "equivalence classes" (schedulable actions with the same label) can change dynamically, because the same action can be associated to different labels during the execution. However we do not know at the moment whether this difference determines a separation in the expressive power.

1.3. Plan of the paper

In the next section we briefly recall some basic notions. In Section 3 we define CCS_σ, a variant of CCS with explicit scheduler. In Section 4 we compare our notion of scheduler with the more standard "semantic" notion. In Section 5 we study the probabilistic testing preorders, their compositionality properties, and the conditions under which (1) holds. Section 6 presents an application to security. Section 7 concludes.

2. Preliminaries

In this section we briefly recall some preliminary notions about simple probabilistic automata, probabilistic bisimulation and CCS_p.

2.1. Simple probabilistic automata ([4, 7])

A *discrete probability space* is a tuple (Ω, μ), where Ω is a countable set and μ is a *discrete probability measure* over Ω, that is, a function $\mu : 2^\Omega \mapsto [0, 1]$ such that $\mu(\Omega) = 1$ and $\mu(\cup_i C_i) = \sum_i \mu(C_i)$ where C_i is a countable family of

pairwise disjoint subsets of Ω. It is also useful to consider discrete probability spaces over an uncountable set Ω', by simply restricting to its countable subsets. Thus, for any set Ω' we denote by $Prob(\Omega')$ the set of all discrete probability spaces (Ω, μ) where Ω is a countable subset of Ω'.

The *Dirac measure* on $x \in \Omega$, denoted by $\delta(x)$, is the probability measure that assigns probability 1 to $\{x\}$. We also denote by $\sum_i [p_i]\mu_i$ the probability measure obtained as a convex sum of the measures μ_i.

A *simple probabilistic automaton*[2] is a tuple $(S, q, A, \mathcal{D})$ where S is a set of states, $q \in S$ is the *initial state*, A is a set of actions and $\mathcal{D} \subseteq S \times A \times Prob(S)$ is a *transition relation*. Intuitively, if $(s, a, (S', \mu)) \in \mathcal{D}$ then there is a transition from the state s performing the action a and leading to a distribution μ over the states S' of the automaton. The idea is that the choice of transition among the available ones in $\mathcal{D}$ is performed nondeterministically, and the choice of the target state among the ones allowed by μ (i.e. those states s such that $\mu(s) > 0$) is performed probabilistically. For simplicity, we omit the sample space S' when this does not create confusion, and we write $s \xrightarrow{a} \mu$ when $(s, a, (S', \mu)) \in \mathcal{D}$.

Note also that the set of states S can be uncountable, but each transition can only reach a countable subset S' of states. In this paper we use automata with an uncountable state space, but whose transitions have only a finite support set.

A probabilistic automaton M is *fully probabilistic* if from each state of M there is at most one transition available. A (partial or complete) execution φ of a probabilistic automaton is a (possibly infinite) sequence $s_0 a_1 s_1 a_2 s_2 \ldots$ of alternating states and actions, such that $q = s_0$, and for each i $(s_i, a_{i+1}, \mu_i) \in \mathcal{D}$ and $\mu_i(s_{i+1}) > 0$ hold. We use $lstate(\varphi)$ to denote the last state of a finite execution φ, and $exec^*(M), exec(M)$ to represent the set of all the finite and of all the executions of M, respectively.

A *scheduler* of a probabilistic automaton $M = (S, q, A, \mathcal{D})$ is a function

$$\zeta : exec^*(M) \mapsto \mathcal{D}$$

such that $\zeta(\varphi) = (s, a, \mu) \in \mathcal{D}$ implies that $s = lstate(\varphi)$. The idea is that a scheduler selects a transition among the ones available in $\mathcal{D}$ and it can base its decision on the history of the execution. The *execution tree* of M relative to the scheduler ζ, denoted by $etree(M, \zeta)$, is a fully probabilistic automaton $M' = (S', q', A', \mathcal{D}')$ such that $S' \subseteq exec^*(M)$, $q' = q$, $A' = A$, and $(\varphi, a, \mu') \in \mathcal{D}'$ if and only if $\zeta(\varphi) = (lstate(\varphi), a, \mu)$ for some μ and $\mu'(\varphi a s) = \mu(s)$. Intuitively, $etree(M, \zeta)$ is produced by unfolding the executions of M and resolving all nondeterministic choices using ζ. Note that $etree(M, \zeta)$ is a simple[3] and fully probabilistic automaton.

[2] For simplicity, in the following we refer to a simple probabilistic automaton as a *probabilistic automaton*. Note however that simple probabilistic automata are a subset of the probabilistic automata defined in [4, 5] which allow a single probabilistic transition to contain multiple distinct actions.

[3] This is true because we do not consider probabilistic schedulers. If we considered such schedulers then the execution tree would no longer be a simple automaton.

We define the probability of a finite execution $\varphi = s_0 a_1 s_1 \dots a_n s_n$ as $pb(\varphi) = \prod_{i=0}^{n-1} \mu_i(s_{i+1})$ where $s_i \xrightarrow{a_{i+1}} \mu_i$, $0 \leq i \leq n-1$. A *cone* with prefix φ is defined as $C(\varphi) = \{\varphi' \in exec^*(M,\zeta) \mid \varphi \leq \varphi'\}$ where $\leq$ is the prefix relation on executions. We define the probability of a cone as $pb(C(\varphi)) = pb(\varphi)$. This way we can construct a probability space on the states of $etree(M,\zeta)$ which allows us to define the probability of any event that can be expressed as a countable union of disjoint cones. More information about the construction can be found in [4].

2.2. Probabilistic bisimulation

If $\mathcal{R}$ is an equivalence relation over a set S, then we can lift it to a relation on probability distributions over S by considering two distributions related if they assign the same probability to the same equivalence classes. More formally two distributions μ_1, μ_2 are equivalent, written $\mu_1 \,\mathcal{R}\, \mu_2$, iff for all equivalence classes $\mathcal{E} \in S/\mathcal{R} : \mu_1(\mathcal{E}) = \mu_2(\mathcal{E})$.

Let $(S, q, A, \mathcal{D})$ be a probabilistic automaton. A *symmetric* relation $\mathcal{R} \subseteq S \times S$ is a *strong bisimulation* iff for all $(s_1, s_2) \in \mathcal{R}$ and for all $a \in A$:

$$s_1 \xrightarrow{a} \mu_1 \;\Rightarrow\; \exists \mu_2 : s_2 \xrightarrow{a} \mu_2 \;\text{ and }\; \mu_1 \,\mathcal{R}\, \mu_2$$

We write $s_1 \sim s_2$ if there is a strong bisimulation that relates them.

The union of two probabilistic automata $M_1 = (S_1, q_1, A, \mathcal{D}_1)$, $M_2 = (S_2, q_2, A, \mathcal{D}_2)$ is an automaton $M = (S_1 \cup S_2, q, A, \mathcal{D}_1 \cup \mathcal{D}_2)$ where $q \in S_1 \cup S_2$. We say that M_1, M_2 are bisimilar if $q_1 \sim q_2$ in M.

2.3. CCS with internal probabilistic choice

Let a range over a countable set of *channel names*. We denote by $\overline{a}$ the corresponding co-action and we generally assume that $\overline{\overline{a}} = a$. The syntax of CCS_p is the following:

$\alpha ::= a \mid \overline{a} \mid \tau$	**prefixes**
$P, Q ::=$	**processes**
$\alpha.P$	prefix
$\mid P \mid Q$	parallel
$\mid P + Q$	nondeterministic choice
$\mid \sum_i p_i P_i$	internal probabilistic choice
$\mid (\nu a)P$	restriction
$\mid\, !a.P$	replicated input
$\mid 0$	nil

with the additional requirement that $\sum_i p_i = 1$ for all probabilistic choices. We also use the notation $P_1 +_p P_2$ to represent a binary probabilistic choice $\sum_{i=1}^{2} p_i P_i$ with $p_1 = p$ and $p_2 = 1 - p$. We denote by $\mathcal{P}_p$ the set of all CCS_σ processes.

$$\text{ACT}\quad \frac{}{\alpha.P \xrightarrow{\alpha} \delta(P)} \qquad \text{RES}\quad \frac{P \xrightarrow{\alpha} \mu \qquad \alpha \neq a, \overline{a}}{(\nu a)P \xrightarrow{\alpha} (\nu a)\mu}$$

$$\text{SUM1}\quad \frac{P \xrightarrow{\alpha} \mu}{P + Q \xrightarrow{\alpha} \mu} \qquad \text{PAR1}\quad \frac{P \xrightarrow{\alpha} \mu}{P \mid Q \xrightarrow{\alpha} \mu \mid Q}$$

$$\text{PROB}\quad \frac{}{\sum_i p_i P_i \xrightarrow{\tau} \sum_i [p_i]\delta(P_i)} \qquad \text{REP}\quad \frac{}{!a.P \xrightarrow{a} \delta(P \mid !a.P)}$$

$$\text{COM}\quad \frac{P \xrightarrow{\beta} \delta(P') \quad Q \xrightarrow{\overline{\beta}} \delta(Q') \quad \beta \in \{a, \overline{a}\}}{P \mid Q \xrightarrow{\tau} \delta(P' \mid Q')}$$

Figure 2: The semantics of CCS_p. SUM1 and PAR1 have corresponding right rules SUM2 and PAR2, omitted for simplicity.

The semantics of a CCS_p term P is a probabilistic automaton $[\![P]\!]$ defined inductively on the basis of the syntax according to the rules in Figure 2. We denote by $\mu \mid Q$ the measure

$$(\mu \mid Q)(X) = \begin{cases} \mu(P) & \text{if } X = P \mid Q \\ 0 & \text{otherwise} \end{cases}$$

Similarly $(\nu a)\mu$ is a measure μ' such that $\mu'((\nu a)P) = \mu(P)$.

A transition of the form $P \xrightarrow{a} \delta(P')$, i.e. a transition having for target a Dirac measure, corresponds to a transition of a non-probabilistic automaton (a standard labeled transition system). Thus, all the rules of CCS_p imitate the ones of CCS except from PROB. The latter models the internal probabilistic choice: a silent τ transition is available from the sum to a measure containing all of its operands, with the corresponding probabilities. Note also that we restrict replication to input prefixes (instead of using full replication or recursion). This greatly simplifies the presentation, while being sufficient for our needs.

Note that in the produced probabilistic automaton, all transitions to non-Dirac measures are silent. This is similar to the *alternating model* ([2]), however our case is more general because the silent and non-silent transitions can be both available at the same state. On the other hand, with respect to the simple probabilistic automata the fact that the probabilistic transitions are silent looks as a restriction. However, it has been proved by Bandini and Segala ([7]) that the simple probabilistic automata and the alternating model are essentially equivalent, so, being in the middle, our model is equivalent as well.

3. A variant of CCS with an explicit scheduler

In this section we present a variant of CCS in which the scheduler is explicit, in the sense that it has a specific syntax and its behavior is defined by the operational semantics of the calculus. We will refer to this calculus as CCS_σ.

$\theta ::= 0\,\theta \mid 1\,\theta \mid \epsilon$	**label indexes**	$S, T ::=$	**scheduler**
$l ::= \kappa^{\theta}$	**labels**	$l.S$	single action
$P, Q ::=$	**processes**	$\mid (l, l).S$	synchronization
$l{:}\alpha.P$	prefix	$\mid$ **if** l	label test
$\mid P \mid Q$	parallel	**then** S	
$\mid P + Q$	nondeterm. choice	**else** S	
$\mid l{:}\sum_i p_i P_i$	prob. choice	$\mid 0$	nil
$\mid (\nu a)P$	restriction	$CP ::= P \parallel S$	**complete proc.**
$\mid\ !l{:}a.P$	replicated input		
$\mid l{:}0$	nil		

Figure 3: The syntax of CCS_σ

Processes in CCS_σ contain labels that allow us to refer to a particular sub-process. A scheduler also behaves like a process, using however a different and much simpler syntax, and its purpose is to guide the execution of the main process using the labels that the latter provides. A *complete process* is a process running in parallel with a scheduler, and we will formally describe their interaction by defining an operational semantics for complete processes.

3.1. Syntax

Let a range over a countable set of *channel names* and κ over a countable set of *base labels*. The syntax of CCS_σ, shown in Figure 3, is the same as the one of CCS_p except for the presence of labels. These are used to select the subprocess which "performs" a transition. Since only the operators with a rule without premises can originate a transition, we only need to assign labels to the prefix, the probabilistic sum and the replicated input. For reasons explained later, we also put labels on 0, even though this is not required for scheduling transitions. We use labels of the form κ^{θ} where κ is a base label and the index θ is a finite string of 0 and 1, possibly empty. Indexes are used to avoid multiple copies of the same label in case of replication, which occurs dynamically due to the bang operator. As explained in the semantics, each time a process is replicated (i.e. a new parallel copy is created) we relabel both sides of the new parallel composition by appending 0 to the labels in the left component, and 1 to those in the right one. For simplicity we write κ for κ^{ϵ} and we use $l_1, l_2, \ldots$ in a process to denote an arbitrary label, with or without an index. The idea of using 0 and 1 to distinguish parallel component positions has been already proposed in the literature, notably by Boudol and Castellani ([20]) and by Bodei, Degano and Priami ([21]).

A scheduler selects a sub-process for execution on the basis of its label, so we use $l.S$ to represent a scheduler that selects the process with label l and continues as S. In the case of synchronization we need to select two processes simultaneously, hence we need a scheduler of the form $(l_1, l_2).S$. Using **if-then-else** the scheduler can test whether a label is available in the process (in the

$$\text{ACT}\quad \frac{}{l{:}\alpha.P \parallel l.S \xrightarrow{\alpha} \delta(P) \parallel S} \qquad \text{RES}\quad \frac{P \parallel S_b \xrightarrow{\alpha} \mu \parallel S \quad \alpha \neq a, \overline{a}}{(\nu a)P \parallel S_b \xrightarrow{\alpha} (\nu a)\mu \parallel S}$$

$$\text{SUM1}\quad \frac{P \parallel S_b \xrightarrow{\alpha} \mu \parallel S}{P + Q \parallel S_b \xrightarrow{\alpha} \mu \parallel S} \qquad \text{PAR1}\quad \frac{P \parallel S_b \xrightarrow{\alpha} \mu \parallel S}{P \mid Q \parallel S_b \xrightarrow{\alpha} \mu \mid Q \parallel S}$$

$$\text{COM}\quad \frac{P \parallel l_1 \xrightarrow{\beta} \delta(P') \parallel 0 \qquad Q \parallel l_2 \xrightarrow{\overline{\beta}} \delta(Q') \parallel 0 \qquad \beta \in \{a, \overline{a}\}}{P \mid Q \parallel (l_1, l_2).S \xrightarrow{\tau} \delta(P' \mid Q') \parallel S}$$

$$\text{PROB}\quad \frac{}{l{:}\sum_i p_i P_i \parallel l.S \xrightarrow{\tau} (\sum_i [p_i]\delta(P_i)) \parallel S}$$

$$\text{REP}\quad \frac{}{!l{:}a.P \parallel l.S \xrightarrow{a} \delta(\rho_0 P \mid \rho_1 !l{:}a.P) \parallel S}$$

$$\text{IF1}\quad \frac{l \in tl(P) \qquad P \parallel S_1 \xrightarrow{\alpha} \mu \parallel S_1'}{P \parallel \textbf{if } l \textbf{ then } S_1 \textbf{ else } S_2 \xrightarrow{\alpha} \mu \parallel S_1'}$$

$$\text{IF2}\quad \frac{l \notin tl(P) \qquad P \parallel S_2 \xrightarrow{\alpha} \mu \parallel S_2'}{P \parallel \textbf{if } l \textbf{ then } S_1 \textbf{ else } S_2 \xrightarrow{\alpha} \mu \parallel S_2'}$$

Figure 4: The semantics of CCS_σ. SUM1 and PAR1 have corresponding right rules SUM2 and PAR2, omitted for simplicity.

top-level) and act accordingly. A scheduler of the form $l.S$, $(l_1, l_2).S$ or 0, that is a scheduler not containing an **if-then-else** at the first step (but it could contain one in the continuation), is called a *base scheduler*. We use S_b to denote base schedulers.

The grammar in Figure 3 defines the set of finite schedulers. We identify each scheduler with its parsing tree, so a scheduler can be viewed as a tree with 0 as leaves and with three types of internal nodes: l-nodes (labelled by a single label), (l, l)-nodes and **if**-nodes. This allows us to extend schedulers to infinite ones, defined as above but on infinite trees. Such schedulers will be used for processes with infinite behaviour. Note that the set of all infinite schedulers is uncountable.

Finally, a complete process is a process put in parallel with a scheduler, for example $l_1 : a.l_2 : b.l_3 : 0 \parallel l_1.l_2.0$. We denote by $\mathcal{P}_\sigma, \mathcal{CP}_\sigma$ the sets of all CCS_σ processes and complete processes respectively.

3.2. Semantics

The operational semantics of the CCS_σ-calculus is given in terms of probabilistic automata defined according to the rules shown in Figure 4. The states of the automaton are complete processes and transitions result to a probability measure on complete processes. Note that for any transition in the semantics,

the complete processes in the support of the resulting measure have all the same scheduler. To make this clear, if $\mu \in Prob(\mathcal{P}_\sigma)$, we denote by $\mu \parallel S \in Prob(\mathcal{CP}_\sigma)$ its lifting to complete processes under scheduler S, that is a measure such that

$$(\mu \parallel S)(X) = \begin{cases} \mu(P) & \text{if } X = P \parallel S \\ 0 & \text{otherwise} \end{cases}$$

Then transitions are of the form $P \parallel S \xrightarrow{\alpha} \mu \parallel S'$ where $\mu \in Prob(\mathcal{P}_\sigma)$.

ACT is the basic communication rule. In order for $l{:}\alpha.P$ to perform α, the scheduler should select this process for execution, so the scheduler needs to be of the form $l.S$. After the execution the complete process will continue as $P \parallel S$. The RES rule models restriction on channel a: communication on this channel is not allowed by the restricted process. SUM1 models nondeterministic choice. If $P \parallel S_b$ can perform a transition, which means that S_b selects one of the labels of P, then $P + Q \parallel S_b$ will perform the same transition, i.e. the branch P of the choice will be selected and Q will be discarded. For example

$$l_1{:}a.P + l_2{:}b.Q \parallel l_1.S \xrightarrow{a} \delta(P) \parallel S$$

Note that the operands of the sum do not have labels, the labels belong to the subprocesses of P and Q. In the case of nested choices, the scheduler must go deep and select the label of a prefix, thus resolving all the choices at once. Also note that we require a base scheduler for this rule. A scheduler starting with **if-then-else** is handled by the IF1,IF2 rules, explained later in this section.

PAR1 has a similar behavior for the parallel composition. The scheduler selects P to perform a transition on the basis of the label. The difference is that in this case Q is not discarded; it remains in the continuation. COM models synchronization. If $P \parallel l_1$ can perform the action a and $Q \parallel l_2$ can perform $\bar{a}$, then $(l_1, l_2).S$, scheduling both l_1 and l_2 at the same time, can synchronize the two. PROB models internal probabilistic choice. Note that the scheduler cannot affect the outcome of the choice, it can only schedule the choice as a whole (this is why a probabilistic sum has a label) and the process will move to a measure containing all the operands with corresponding probabilities.

REP models replicated input. This rule is the same as for CCS_p, with the addition of a re-labeling operator ρ_i. The reason for this is that we want to avoid ending up with multiple copies of the same label as the result of replication, since this would create ambiguities in scheduling as explained in Section 3.3. $\rho_i P$ appends $i \in \{0, 1\}$ to the index of all labels of P, for example:

$$\rho_i \kappa^\theta{:}\alpha.P = \kappa^{\theta i}{:}\alpha.\rho_i P$$

and similarly for the other operators. Note that we relabel only the resulting process, not the continuation of the scheduler. There is no need to do so, since the continuation of the scheduler can simply use the labels with the added index, after the replication.

Finally **if-then-else** allows the scheduler to adjust its behavior based on the labels that are available in P. We denote by $tl(P)$ the set of top-level labels of

P, defined as

$$tl(l{:}\alpha.P) = tl(l{:}\textstyle\sum_i p_i P_i) = tl(!l{:}a.P) = tl(l{:}0) = \{l\}$$
$$tl(P+Q) = tl(P \mid Q) = tl(P) \cup tl(Q)$$
$$tl((\nu a)P) = tl(P)$$

Then **if** l **then** S_1 **else** S_2 behaves like S_1 if l is available in P and as S_2 otherwise. This is needed when P is the outcome of a probabilistic choice, as discussed in Section 4.

Note that all rules apart from IF1,IF2 require a base scheduler. In other words, in the proof tree of a transition, all applications of the rules IF1,IF2 must appear at the very end. This is to ensure that the IF1,IF2 rules take all the labels of the process into account. For example, having shown that $l_1{:}a \parallel$ **if** l_2 **then** l_2 **else** $l_1 \xrightarrow{a} \delta(0) \parallel 0$, we cannot use the SUM1 rule to prove the same transition for $l_1 : a + l_2 : b$, because now l_2 is available thus the **if** branch should be selected.

3.3. Deterministic labelings

The idea in CCS_σ is that a *syntactic* scheduler will be able to completely resolve the nondeterminism of the process, without needing to rely on a *semantic* scheduler at the level of the automaton. This means that the execution of a process in parallel with a scheduler should be fully probabilistic. To achieve this we impose a condition on the labels that we can use in CCS_σ processes. A *labeling* is an assignment of labels to the prefixes, the probabilistic sums, the replicated inputs and the 0s of a process. We require all labelings to be *deterministic* in the following sense.

Definition 1. We define the relation $\longrightarrow$ on complete processes as $P \parallel S \longrightarrow P' \parallel S'$ iff $P \parallel S \xrightarrow{\alpha} \mu \parallel S'$ for some α and $\mu(P') > 0$. We define $\longrightarrow^*$ as the reflexive and transitive closure of $\longrightarrow$. With a slight abuse of notation we define the same relation on processes (without schedulers) as $P \longrightarrow P'$ iff $P \parallel S \longrightarrow P' \parallel S'$ for some S, S'.

Definition 2. A labeling of a process P is *deterministic* iff for all schedulers S and for all P', S' such that $P \parallel S \longrightarrow^* P' \parallel S'$ there is at most one transition rule $P' \parallel S' \xrightarrow{\alpha}$ that can be applied.

In the general case, it is impossible to decide whether a particular labeling is deterministic. However, there are simple ways to construct labeling that are guaranteed to be deterministic. A simple such family are the linear labelings.

Definition 3. A labeling is called *linear* iff for all labels $\kappa_1^{\theta_1}, \kappa_2^{\theta_2}$ appearing in the process, $\kappa_1 \neq \kappa_2$ or $\theta_1 \not\preceq \theta_2 \wedge \theta_2 \not\preceq \theta_1$, where $\preceq$ is the prefix relation on indexes.

The idea is that in a linear labeling all labels should be pairwise distinct. The extra condition on the indexes forbids having two (distinct) labels κ, κ^0 since they could become equal as the result of relabeling the first. This is important for the following proposition.

Proposition 1. *Linear labelings are preserved by transitions.*

PROOF. Let $lab(P)$ denote the set of all labels of P. First, note that the transition rules only *append* strings to the indexes of the process' labels. That is, if $P \longrightarrow Q$ and $\kappa^\eta \in lab(Q)$ then there exists a label $\kappa^\theta \in P$ such that $\theta \preceq \eta$. This is clear since the relabeling operator ρ_i only appends strings to indexes.

We write $\theta \nsim \eta$ for $\theta \not\preceq \eta \wedge \eta \not\preceq \theta$. First, we notice that $\theta \nsim \eta$ iff $\theta_i \neq \eta_i$ for some $i \leq \min\{|\theta|, |\eta|\}$ where θ_i, η_i denote the i-th character of θ, η respectively. As a consequence we have that

$$\theta \nsim \eta \;\Rightarrow\; \theta\theta' \nsim \eta\eta' \qquad \text{for all } \theta', \eta' \tag{2}$$

since $\theta\theta'$ and $\eta\eta'$ still differ at the i-th character.

The proof is by induction of the "proof tree" of the transition. For the base case, the rules ACT, PROB are easy since the labels of the resulting process are a subset of the original ones. The interesting rule for the base case is the REP rule:

$$\overline{!l{:}a.P \parallel l.S \xrightarrow{\alpha} \delta(\rho_0 P \mid \rho_1 !l{:}a.P) \parallel S}$$

The labels of the resulting process are of the form $\kappa^{\theta i}$ where $\kappa^\theta \in lab(!l : a.P)$ and $i \in \{0, 1\}$. So consider two such labels $\kappa^{\theta i}, \kappa^{\eta j}$. Since $!l : a.P$ has a linear labeling, we have $\theta \nsim \eta$ and from (2) we get $\theta i \nsim \eta j$.

For the inductive case, the rules RES, SUM1/2, IF1, IF2 are easy since the resulting measure μ is the same as in the premise, so a direct application of the induction hypothesis suffices. Now consider the PAR1 rule

$$\frac{P \parallel S_b \xrightarrow{\alpha} \mu \parallel S}{P \mid Q \parallel S_b \xrightarrow{\alpha} \mu \mid Q \parallel S}$$

Assume that $P \mid Q$ has a linear labeling and consider a process P' such that $\mu(P') > 0$. We want to show that $P' \mid Q$ has a linear labeling, that is, if two labels of $P' \mid Q$ have the same base then their indexes must be prefix-incomparable. Since Q has a linear labeling and so does P' (from the induction hypothesis), we only need to compare indexes between P' and Q. Let $\kappa^\eta \in lab(P'), \kappa^\iota \in lab(Q)$. Since P' comes from a transition of P then there exists $\kappa^\theta \in lab(P)$ such that $\theta \preceq \eta$, and since $P \mid Q$ has a linear labeling then $\theta \nsim \iota$. So from (2) we have $\eta \nsim \iota$. □

Proposition 2. *A linear labeling is deterministic.*

PROOF. Let P be a process with a linear labeling, let S be a scheduler and $P' \parallel S'$ be a complete process such that $P \parallel S \longrightarrow^* P' \parallel S'$. We want to

show that there is only one transition $P' \parallel S' \xrightarrow{\alpha} \mu \parallel S''$ enabled. Since linear labelings are preserved by transitions, P' has also a linear labeling. As a consequence, its labels are pairwise distinct, so the label(s) in the root of S' appear at most once in P'. So from the rules PAR1/PAR2, at most one is applicable, since at most one branch of $P \mid Q$ contains the required label. The same holds for SUM1/SUM2.

We want to show that we can construct at most one proof tree for the transition of $P' \parallel S'$. Since we eliminated one rule of the pairs PAR1/2, SUM1/2, for the remaining rules and for a fixed "type" of process and scheduler, there is at most one rule applicable. For example, for $P \mid Q$ and $l.S$ only PAR is applicable, for $P \mid Q$ and $(l_1, l_2).S$ only COM is applicable and for $!l{:}a.P$ and $l.S$ only REP. For any process and an **if-then-else** scheduler only one of IF1, IF2 is applicable, and so on. Since the premises of all rules involve a simpler process or a simpler scheduler, the result comes easily by induction on the structure of $P' \parallel S'$. □

There are labelings that are deterministic without being linear. In fact, such labelings will be the means by which we hide information from the scheduler. However, the property of being deterministic is crucial since it implies that the scheduler will resolve all the nondeterminism of the process.

Proposition 3. *Let P be a CCS_σ process with a deterministic labeling. Then $[\![P \parallel S]\!]$ is fully probabilistic for all schedulers S.*

Proof. Direct application of the definition of deterministic labeling. □

4. Expressiveness of the syntactic scheduler

CCS_σ with deterministic labelings allows us to separate probabilities from nondeterminism in a straightforward way: a process in parallel with a scheduler behaves in a fully probabilistic way and the nondeterminism arises from the fact that we can have many different schedulers. We may now ask the question: how powerful are the syntactic schedulers wrt the semantic ones, i.e. those defined directly over the automaton?

We denote by $Unlab(\widehat{P})$ the CCS_p process obtained from the CCS_σ process $\widehat{P}$ by removing all labels. Moreover, let

$$Lin(P) = \{\widehat{P} \in \mathcal{CP}_\sigma \mid Unlab(\widehat{P}) = P \text{ and } \widehat{P} \text{ has a linear labeling}\}$$

We say that the semantic scheduler ζ of $[\![P]\!]$ is equivalent to the syntactic scheduler S of $\widehat{P} \in Lin(P)$, written $\zeta \sim_P S$, iff the automata $etree([\![P]\!], \zeta)$ and $[\![\widehat{P} \parallel S]\!]$ are probabilistically bisimilar.

A process is blocked if it cannot perform a transition under any scheduler. A scheduler S is *non-blocking* for a process P if it always schedules some transition, except when P itself is blocked. Let $Sem(P)$ be the set of the semantic schedulers for the process P and $Syn(\widehat{P})$ be the set of the non-blocking syntactic schedulers for process $\widehat{P}$. Then we can show that for all semantic schedulers of P we can create an equivalent syntactic one for $\widehat{P}$.

Proposition 4. *Let P be a CCS$_p$ process and let $\widehat{P} \in Lin(P)$. Then $\forall \zeta \in Sem(P)\ \exists S \in Syn(\widehat{P}) : \zeta \sim_P S$.*

Proof. We fix the processes P_0 and $\widehat{P}_0 \in Lin(P_0)$ for which we are going to prove the proposition, and let $M = (S, P_0, A, \mathcal{D}) = [\![P_0]\!]$. We also fix a scheduler $\zeta : exec^*(M) \to \mathcal{D}$ for M. An execution $\varphi \in exec^*(M)$ of M is a sequence $\varphi = P_0\alpha_1 P_1 \ldots \alpha_n P_n$ such that $P_{i-1} \xrightarrow{\alpha_i} \mu$ and $\mu(P_i) > 0$. Then $etree(M, \zeta)$ is a fully probabilistic automaton whose set of states is $exec^*(M)$ and where $\varphi \xrightarrow{\alpha} \mu_e$ iff $\zeta(\varphi) = (P_n, \alpha, \mu)$ and $\mu_e(\varphi\alpha P_{n+1}) = \mu(P_{n+1})$. We aim at finding a syntactic scheduler S_ζ such that $\zeta \sim_{P_0} S_\zeta$.

First note that for each rule in the semantics of CCS$_p$ there is a corresponding rule for CCS$_\sigma$, the only addition being the syntactic scheduler and the labels of the resulting processes. Thus, we can show (by induction on the proof tree of $P \xrightarrow{\alpha} \mu$) that for all $P, \widehat{P}$ with $\widehat{P} \in Lin(P)$:

$$\begin{aligned} P \xrightarrow{\alpha} \mu \quad \Rightarrow \quad & \exists S : \widehat{P} \parallel S \xrightarrow{\alpha} \mu' \parallel 0 \text{ and} \\ & \forall P' \in \mathcal{P}_p \ : \mu(P') = \mu'(Lin(P')) \end{aligned} \tag{3}$$

The scheduler S above has no continuation, since it is reduced to 0 after the transition. There might still be several schedulers producing this transition, but it is easy to see that there exists a unique minimal one, i.e. a scheduler of the form $l.0$ or $(l_1, l_2).0$. If $t = (P, \alpha, \mu) \in \mathcal{D}$ is the tuple describing the transition of P, let $sched(t, \widehat{P})$ denote this minimal scheduler. Note also that a single process in μ might be mapped to several CCS$_\sigma$ processes in μ' that only differ in the labels. For example, $a +_p a \xrightarrow{\tau} \delta(a)$, but after adding a linear labeling we have $l{:}(l_1{:}a +_p l_2{:}a) \parallel l.0 \xrightarrow{\tau} \frac{1}{2}\delta(l_1{:}a) + \frac{1}{2}\delta(l_2{:}a)$.

Now, given $P \in \mathcal{P}_p$, a state $\varphi \in exec^*(M)$ such that $lstate(\varphi) = P$ and a process $\widehat{P} \in Lin(P)$, we construct the syntactic scheduler $\sigma(\zeta, \varphi, \widehat{P}) \in Syn(\widehat{P})$ corresponding to ζ. Let $(P, \alpha, \mu) = \zeta(\varphi)$. From (3) there is a transition $\widehat{P} \parallel S \xrightarrow{\alpha} \mu' \parallel 0$ with $S = sched(\zeta(\varphi), \widehat{P})$. Let $\{\widehat{P}_1, \ldots, \widehat{P}_n\} = \text{support}(\mu')$ and $P_i = Unlab(\widehat{P}_i)$, $1 \le i \le n$. We denote by $lm(\widehat{P})$ the left-most label appearing in $\widehat{P}$. Note that all processes contain at least one label since they contain at least one 0. We recursively define $\sigma(\zeta, \varphi, \widehat{P})$ as

$$\begin{aligned} \sigma(\zeta, \varphi, \widehat{P}) \triangleq\ & sched(\zeta(\varphi), \widehat{P}). \\ & \quad \textbf{if } lm(\widehat{P}_1) \textbf{ then } \sigma(\zeta, \varphi\alpha P_1, \widehat{P}_1) \textbf{ else} \\ & \quad \ldots \\ & \quad \textbf{if } lm(\widehat{P}_{n-1}) \textbf{ then } \sigma(\zeta, \varphi\alpha P_{n-1}, \widehat{P}_{n-1}) \textbf{ else} \\ & \quad \sigma(\zeta, \varphi\alpha P_n, \widehat{P}_n) \end{aligned} \tag{4}$$

We then define an equivalence $\equiv_P$ on schedulers as

$$S_1 \equiv_P S_2 \quad \text{iff} \quad P \parallel S_1 \xrightarrow{\alpha} \mu \parallel S_3 \ \Leftrightarrow\ P \parallel S_2 \xrightarrow{\alpha} \mu \parallel S_3$$

Intuitively $S_1 \equiv_P S_2$ iff they have the same effect on the process P, for example if S_1 is an **if-then-else** construct that enables S_2.

We are now ready for the final part of the proof. Let $\varphi_0 = P_0$ be an empty execution of M, the scheduler S_ζ that we want to construct is $S_\zeta = \sigma(\zeta, \varphi_0, \widehat{P}_0)$. We compare the automata $etree(M, \zeta)$ and $[\![\widehat{P}_0 \parallel S_\zeta]\!]$ and we show that they are bisimilar by creating a bisimulation relation that relates their starting states φ_0 and $\widehat{P}_0 \parallel S_\zeta$. We define a relation $\mathcal{R} \subseteq exec^*(M) \times \mathcal{CP}_\sigma$ as follows:

$$\varphi \; \mathcal{R} \; (\widehat{P} \parallel S) \quad \text{iff} \quad lstate(\varphi) = Unlab(\widehat{P}) \;\text{ and }\; S \equiv_{\widehat{P}} \sigma(\zeta, \varphi, \widehat{P})$$

Clearly $\varphi_0 \; \mathcal{R} \; (\widehat{P}_0 \parallel S_\zeta)$ and, hence, it remains to show that $\mathcal{R}$ is a strong bisimulation.

Suppose that $\varphi \; \mathcal{R} \; (\widehat{P} \parallel S)$ and $\varphi \xrightarrow{\alpha} \mu_e$. Let $P = Unlab(\widehat{P})$, since $lstate(\varphi) = P$ we have $P \xrightarrow{\alpha} \mu$ with $\mu(P') = \mu_e(\varphi\alpha P')$ for all $P' \in \mathcal{P}_p$. From $S \equiv_{\widehat{P}} \sigma(\zeta, \varphi, \widehat{P})$ and by construction of σ we have that there exists a transition $\widehat{P} \parallel S \xrightarrow{\alpha} \mu' \parallel S_c$ where $\mu'(Lin(P')) = \mu(P') = \mu_e(\varphi\alpha P')$ for all $P' \in \mathcal{P}_p$.

Let $\{\widehat{P}_1, \ldots, \widehat{P}_n\} = \text{support}(\mu')$ and $P_i = Unlab(\widehat{P}_i), 1 \le i \le n$ (note that we might have $P_i = P_j$ for $i \neq j$). The scheduler S_c above is the continuation of $\sigma(\zeta, \varphi, \widehat{P})$ (defined in (4)). It is an **if-then-else** choice between the schedulers $\sigma(\zeta, \varphi\alpha P_i, \widehat{P}_i)$, each guarded by **if** $lm(\widehat{P}_i)$. Since the labeling of $\widehat{P}$ is linear, all labels are pairwise distinct, so the $\widehat{P}_i$'s have disjoint labels, i.e. $lm(\widehat{P}_i) \notin tl(\widehat{P}_j)$ for $i \neq j$. As a consequence, $S_c \equiv_{\widehat{P}_i} \sigma(\zeta, \varphi\alpha P_i, \widehat{P}_i)$ since only the i-th branch of S_c can be enabled by $\widehat{P}_i$. Thus we have $(\varphi\alpha P_i) \; \mathcal{R} \; (\widehat{P}_i \parallel S_c)$, for all $1 \le i \le n$, which, from $\mu_e(\varphi\alpha P_i) = \mu'(Lin(P_i))$, implies that $\mu_e \; \mathcal{R} \; (\mu' \parallel S_c)$.

Similarly for the case where $\widehat{P} \parallel S \xrightarrow{\alpha} \mu' \parallel S_c$. Let $\{\widehat{P}_1, \ldots, \widehat{P}_n\} = \text{support}(\mu')$ and $P_i = Unlab(\widehat{P}_i), 1 \le i \le n$. By definition of $\sigma(\zeta, \varphi, \widehat{P})$ there exists a transition $P \xrightarrow{\alpha} \mu$ where $\mu(P_i) = \mu'(Lin(P_i))$, thus $\varphi \xrightarrow{\alpha} \mu_e$ with $\mu_e(\varphi\alpha P_i) = \mu'(Lin(P_i))$. So again $(\varphi\alpha P_i) \; \mathcal{R} \; (\widehat{P}_i \parallel S_c)$, for all $1 \le i \le n$, thus $\mu_e \; \mathcal{R} \; (\mu' \parallel S_c)$. □

To obtain this result the label test (**if-then-else**) is crucial, in the case P performs a probabilistic choice. The scheduler uses the test to find out the result of the probabilistic choice and adapt its behavior accordingly (as the semantic scheduler is allowed to do). For example let $P = l\!:\!(l_1\!:\!a +_p l_2 : b) \mid (l_3\!:\!c + l_4\!:\!d)$. For this process, the scheduler $l.$(**if** l_1 **then** $l_3.l_1$ **else** $l_4.l_2$) first performs the probabilistic choice. If the result is $l_1\!:\!a$ it performs c, a, otherwise it performs d, b. This is also the reason we need labels for 0, in case it is one of the operands of the probabilistic choice.

One would expect to obtain also the inverse of Proposition 4, showing the same expressive power for the two kinds of schedulers. We believe that this is indeed true, but it is technically more difficult to state. The reason is that the simple translation we did from CCS_p processes to CCS_σ, namely adding a linear labeling, might introduce choices that are not present in the original process.

For example let $P = (a +_p a) \mid (c + d)$ and $\widehat{P} = l{:}(l_1{:}a +_p l_2 : a) \mid (l_3{:}c + l_4{:}d)$. In P the choice $a +_p a$ is not a real choice, it can only do an τ transition and go to a with probability 1. But in $\widehat{P}$ we make the two outcomes distinct due to the labeling. So the syntactic scheduler $l.(\textbf{if } l_1 \textbf{ then } l_3.l_1 \textbf{ else } l_4.l_2)$ has no semantic counterpart simply because $\widehat{P}$ has more choices than P, but this is an artifact of the translation. A more precise translation that would establish the exact correspondence of schedulers is left as future work.

4.1. Using non-linear labelings

Up to now we are using only linear labelings which, as we saw, give us the whole power of semantic schedulers. However, we can construct non-linear labelings that are still deterministic, that is there is still only one transition possible at any time even though we have multiple occurrences of the same label. There are various cases of useful non-linear labelings.

Proposition 5. *Let P,Q be CCS_σ processes with deterministic labelings (not necessarily disjoint). The following labelings are all deterministic:*

$$l{:}(P +_p Q) \tag{5}$$

$$l_1{:}a.P + l_2{:}b.Q \tag{6}$$

$$(\nu a)(\nu b)(l_1{:}a.P + l_1{:}b.Q \mid l_2{:}\bar{a}) \tag{7}$$

Proof. Processes (5),(7) have only one transition enabled, while (6) has two, all enabled by exactly one scheduler. After any of these transitions, only one of P, Q remains. □

Consider the case where P and Q in the above proposition share the same labels. In (5) the scheduler cannot select an action inside P, Q, it must select the choice itself. After the choice, only one of P, Q will be available so there will be no ambiguity in selecting transitions. The case (6) is similar but with nondeterministic choice. Now the guarding prefixes must have different labels, since the scheduler should be able to resolve the choice, however after the choice only one of P, Q will be available. Hence, again, the multiple copies of the labels do not constitute a problem. In (7) we allow the same label on the guarding prefixes of a nondeterministic choice. This is because the guarding channels a, b are restricted and only one of the corresponding output actions is available ($\bar{a}$). As a consequence, there is no ambiguity in selecting transitions. A scheduler (l_1, l_2) can only perform a synchronization on a, even though l_1 appears twice.

However, using multiple copies of a label limits the power of the scheduler, since the labels provide information about the outcome of a probabilistic choice (and allow the scheduler to choose different strategies through the use of the scheduler choice). In fact, this is exactly the technique we use to achieve the goals described in Section 1. Consider for example the process:

$$l{:}(l_1{:}\bar{a}.R_1 +_p l_1{:}\bar{a}.R_2) \mid l_2{:}a.P \mid l_3{:}a.Q \tag{8}$$

According to Proposition 5 (5) this labeling is deterministic. However, since both branches of the probabilistic sum have the same label l_1, the scheduler cannot resolve the choice between P and Q based on the outcome of the probabilistic choice. There is still nondeterminism: the scheduler $l.(l_1, l_2)$ will select P and the scheduler $l.(l_1, l_3)$ will select Q. However this selection will be independent from the outcome of the probabilistic choice.

Note that we did not impose any direct restrictions on the schedulers. We still consider all possible syntactic schedulers for the process (8) above. However, having the same label twice limits the power of the syntactic schedulers with respect to the semantic ones. This approach has the advantage that the restrictions are limited to the choices with the same label. We already know that having pairwise distinct labels gives the full power of the semantic scheduler. So the restriction is local to the place where we, intentionally, put the same labels.

5. Testing relations for CCS_σ processes

Testing relations ([22]) are a method of comparing processes by considering their interaction with the environment. A *test* is a process that runs in parallel with the one being tested, and that contains a distinguished action ω that represents success. Two processes are testing equivalent if they can pass the same tests. This idea is very useful for the analysis of security protocols, as suggested in [23], since a test can be seen as an adversary who interferes with a communication agent and declares ω if an attack is successful. Then two processes are testing equivalent if they are vulnerable to the same attacks.

In the probabilistic setting we take the approach of [13] which considers the exact probability of passing a test, in contrast to [10] which considers only the ability to pass a test with probability non-zero (may-testing) or one (must-testing). This approach leads to the definition of two preorders $\sqsubseteq_{\mathbf{may}}$ and $\sqsubseteq_{\mathbf{must}}$. Intuitively, $P \sqsubseteq_{\mathbf{may}} Q$ means that if P under some scheduler passes O with probability p then Q also passes O under some scheduler with at least the same probability. $P \sqsubseteq_{\mathbf{must}} Q$ means that if P under any scheduler passes O with probability at least p then Q under all schedulers passes O with at least the same probability.

More precisely, a test O is a CCS_σ process containing the distinguished action ω, such that when put in parallel with any of the tested processes, the resulting labeling is deterministic. Let $Test_{\mathcal{P}}$ denote the set of all tests with respect to the set of processes $\mathcal{P}$ and let $(\nu)P$ denote the restriction on all channels of P, thus allowing only τ actions. We define $p_\omega(P, S, O)$ to be the probability that $(\nu)(P \mid O) \parallel S$ produces ω:

$$p_\omega(P, S, O) = pb\Big(\bigcup\{C(\varphi\omega s) \mid \varphi\omega s \in exec^*([\![(\nu)(P \mid O) \parallel S]\!])\}\Big)$$

Note that the set on the right hand side is a countable union of disjoint cones so its probability is well-defined. We can now define may and must testing.

Definition 4. Let P, Q be CCS_σ processes. We define the must and may testing preorders as follows:

$$P \sqsubseteq_{\mathbf{may}} Q \quad \text{iff} \quad \forall O\ \forall S_P\ \exists S_Q\ :\ p_\omega(P, S_P, O) \leq p_\omega(Q, S_Q, O)$$
$$P \sqsubseteq_{\mathbf{must}} Q \quad \text{iff} \quad \forall O\ \forall S_Q\ \exists S_P\ :\ p_\omega(P, S_P, O) \leq p_\omega(Q, S_Q, O)$$

where O ranges over $Test_{\{P,Q\}}$ and S_X ranges over $Syn((\nu)(X \mid O))$.

We also define $\approx_{\mathbf{may}}, \approx_{\mathbf{must}}$ to be the equivalences induced by $\sqsubseteq_{\mathbf{may}}, \sqsubseteq_{\mathbf{must}}$ respectively.

Note that Definition 4 is slightly different than the definition of probabilistic testing of [13], given below:

Definition 5. Let P, Q be CCS_σ processes and O a test. We first define:

$$P\lceil O \rceil = \sup\{p_\omega(P, S, O) \mid S \in Syn((\nu)(P \mid O))\}$$
$$P\lfloor O \rfloor = \inf\{p_\omega(P, S, O) \mid S \in Syn((\nu)(P \mid O))\}$$

The may and must testing preorders of [13] are defined as:

$$P \leq_{\mathbf{may}} Q \quad \text{iff} \quad P\lceil O \rceil \leq Q\lceil O \rceil \quad \forall O \in Test_{\{P,Q\}}$$
$$P \leq_{\mathbf{must}} Q \quad \text{iff} \quad P\lfloor O \rfloor \leq Q\lfloor O \rfloor \quad \forall O \in Test_{\{P,Q\}}$$

The above definition is arguably closer to the informal intuition and easier to understand. However, the use of $\sup, \inf$ in $P\lceil O \rceil, P\lfloor O \rfloor$ makes it difficult to use in proofs. Instead, we use Definition 4 which turns out to be slightly stronger.

Proposition 6. *For all CCS_σ processes P, Q:*

$$P \sqsubseteq_{may} Q \ \Rightarrow\ P \leq_{may} Q$$
$$P \sqsubseteq_{must} Q \ \Rightarrow\ P \leq_{must} Q$$

The inverse is also true for finite processes.

Proof. We use the simple fact that for any non-empty sets $A, B \subseteq \mathbb{R}$:

$$\forall a \in A\ \exists b \in B : a \leq b \ \Rightarrow\ \sup A \leq \sup B$$
$$\forall b \in B\ \exists a \in A : a \leq b \ \Rightarrow\ \inf A \leq \inf B$$

Assuming $\sup A > \sup B$, let $k = (\sup A + \sup B)/2$. Since $k < \sup A$ there exists $a \in A$ such that $k < a$. Then there exists $b \in B$ such that $a \leq b$ so we have $\sup B < k < a \leq b$ which is a contradiction. Similarly for inf.

The inverse does not hold in general, for example if $A = [0, 1], B = (0, 1)$ we have $\sup A = \sup B = 1$ but $1 > b\ \forall b \in B$. However it holds if $\sup A \in A, \sup B \in B$, for example if A, B are finite. □

The difference between $\sqsubseteq_{\mathbf{may}}$ and $\leq_{\mathbf{may}}$ lies in cases where the supremum of $p_\omega(P, S, O)$ cannot be achieved by any single scheduler S. For finite processes this is never the case since the set of schedulers is finite.

5.1. Compositionality properties

In this section we study some compositionality properties of the testing preorders for CCS_σ. A context C is a process containing a hole that we denote by $[\,]$. The application of C to a process P, denoted by $C[P]$ is the process obtained by replacing $[\,]$ by P. Note that we can only apply C to P if the labeling of $C[P]$ is deterministic. A preorder $\sqsubseteq$ is a precongruence if $P \sqsubseteq Q$ implies $C[P] \sqsubseteq C[Q]$ for all contexts C. A labeling of a process is *fresh* (with respect to a set $\mathcal{P}$ of processes) if its labels are distinct from the labels of any process in $\mathcal{P}$. Note that a fresh labeling is not necessarily linear. The following proposition states that may testing is a precongruence if we restrict to contexts with fresh labelings, and must testing is also a precongruence if, additionally, we restrict to contexts where the + occurs only guarded. More precisely, this means that if $C = C' + R$ then $C' = a.C''$ for some $a \neq \tau$, and recursively, the same must hold for all subcontexts of C. This result is essentially an adaptation to our framework of the analogous precongruence property in [3][4].

Proposition 7. *Let P, Q be CCS_σ processes such that $P \sqsubseteq_{\mathbf{may}} Q$ and let C be a context with a fresh labeling (wrt P, Q). Then $C[P] \sqsubseteq_{\mathbf{may}} C[Q]$. Similarly for $\sqsubseteq_{\mathbf{must}}$, provided that any occurrence of $+$ in C is guarded. If $+_p$ does not occur in C then the restriction on C's labeling can be dropped.*

Proof. We first prove the proposition for contexts where $[\,]$ is not under replication. Without loss of generality we assume that tests do not perform internal actions, but only synchronizations with the tested process. The proof will be by induction on the structure of C. Let O range over tests, let S_P range over $Syn((\nu)(C[P] \mid O))$ and S_Q range over $Syn((\nu)(C[Q] \mid O))$. The induction hypothesis is:

$$\begin{array}{l}\textbf{may})\ \forall O\ \forall S_P\ \exists S_Q\ :\ p_\omega(C[P], S_P, O) \leq p_\omega(C[Q], S_Q, O) \quad \text{and} \\ \textbf{must})\ \forall O\ \forall S_Q\ \exists S_P\ :\ p_\omega(C[P], S_P, O) \leq p_\omega(C[Q], S_Q, O)\end{array}$$

We have the following cases for C. Note that we use the freshness restriction only for the probabilistic choice.

- Case $C = [\,]$. Trivial.
- Case $C = l_1 : a.C'$
  The scheduler S_P has to be of the form $S_P = (l_1, l_2).S'_P$ where l_2 is the label of a $\overline{a}$ prefix in O (if no such prefix exists then the case is trivial).

  A scheduler of the form $(l_1, l_2).S$ can schedule any process of the form $l_1 : a.X$ (with label l_1) giving the transition:

$$(\nu)(l_1 : a.X \mid O) \parallel (l_1, l_2).S \xrightarrow{\tau} \delta((\nu)(X \mid O')) \parallel S$$

[4] The authors of [3] considered only the case of context without occurrences of $+$, but we believe that our more liberal restriction would have been sufficient also for obtaining the result in [3].

and producing always the same O'. The probability p_ω will be

$$p_\omega(l_1{:}a.X, (l_1, l_2).S, O) = p_\omega(X, S, O') \tag{9}$$

Thus for (**may**) we have

$$\begin{array}{rcll} p_\omega(C[P], (l_1, l_2).S'_P, O) & = & p_\omega(C'[P], S'_P, O') & (9) \\ & \leq & p_\omega(C'[Q], S'_Q, O') & \text{Ind. Hyp.} \\ & = & p_\omega(C[Q], (l_1, l_2).S'_Q, O) & (9) \\ & = & p_\omega(C[Q], S_Q, O) & \end{array}$$

For (**must**) we can perform the above derivation in the opposite direction, given that a scheduler for $C[Q]$ must be of the form $S_Q = (l_1, l_2).S'_Q$.

- Case $C = C' \mid R$
  We have that $R \mid O$ is itself a test and

$$p_\omega(X \mid R, S, O) = p_\omega(X, S, R \mid O) \tag{10}$$

Thus for (**may**) we have

$$\begin{array}{rcll} p_\omega(C[P], S_P, O) & = & p_\omega(C'[P], S_P, R \mid O) & (10) \\ & \leq & p_\omega(C'[Q], S_Q, R \mid O) & \text{Ind. Hyp.} \\ & = & p_\omega(C[Q], S_Q, O) & (10) \end{array}$$

For (**must**) we can perform the above derivation in the opposite direction.

- Case $C = l_1{:}(C' +_p R)$
  Let us first assume that P is in the top-level of $C'[P]$. In order to be non-blocking, the scheduler of $l_1 : (C'[P] +_p R)$ must detect the outcome of the probabilistic choice and continue as S_C if the outcome is $C'[P]$ or as S_R otherwise. For example S_P could be l_1.**if** l **then** S_C **else** S_R or a more complicated **if-then-else**. So we have

$$\begin{array}{r} p_\omega(l_1{:}(C'[P] +_p R), S, O) = p\ p_\omega(C'[P], S_C, O) + \\ \bar{p}\ p_\omega(R, S_R, O) \end{array} \tag{11}$$

where $\bar{p} = 1 - p$. For (**may**) we have

$$\begin{array}{rcll} & & p_\omega(l_1{:}(C'[P] +_p R), S_P, O) & \\ & = & p\ p_\omega(C'[P], S_C, O) + \bar{p}\ p_\omega(R, S_R, O) & (11) \\ & \leq & p\ p_\omega(C'[Q]), S'_C, O) + \bar{p}\ p_\omega(R, S_R, O) & \text{Ind. Hyp.} \\ & = & p_\omega(l_1{:}(C'[Q] +_p R), l_1.(\textbf{if } l \textbf{ then } S'_C \textbf{ else } S_R), O) & \\ & = & p_\omega(C[Q], S_Q, O) & \end{array}$$

Where $l \in tl(Q)$, which means that $l \notin tl(R)$ since we consider only contexts with a fresh labeling. We used the **if-then-else** in S_Q to imitate

the test of S_P and the fact that $l \notin tl(R)$ is crucial. (note that we use the freshness restriction only for the probabilistic choice) If P is not in the top-level of $C'[P]$ then S_P will have the same behaviour on $C'[Q]$. So we can construct the scheduler S_Q by duplicating S_P until the point where P becomes top-level, when the previous case applies.

For (**must**) we can perform the above derivation in the opposite direction.

- Case $C = C' + R$
  Let $T(X) = (\nu)(X \mid O)$. We first prove the case of (**may**). Assuming that $T(C[P])$ is not blocked (the other case is trivial), a scheduler S_P for $T(C[P])$ has to choose between $C'[P]$ and R, using the rules SUM1 and SUM2 respectively. Let us consider the two cases:

  - The transition of $T(C[P])$ is obtained using SUM1. In this case

    $$p_\omega(C'[P] + R, S_P, O) = p_\omega(C'[P], S_P, O)$$

    From the Ind. Hyp., there exists a scheduler S_Q for $T(C'[Q])$ s.t.

    $$p_\omega(C'[P], S_P, O) \leq p_\omega(C'[Q], S_Q, O)$$

    If $T(C'[Q])$ is not blocked then S_Q is a non-blocking scheduler also for $T(C[Q])$, and we have

    $$p_\omega(C'[P], S_P, O) \leq p_\omega(C'[Q], S_Q, O) = p_\omega(C'[Q] + R, S_Q, O)$$

    If $T(C'[Q])$ is blocked then S_Q might not be a valid scheduler for $T(C[Q])$. However in this case $p_\omega(C'[Q], S_Q, O) = p_\omega(C'[P], S_P, O) = 0$ and the result holds trivially for any scheduler of $T(C[Q])$.
  - The transition of $T(C[P])$ is obtained using SUM2. In this case, using the same scheduler S_P for $T(C[Q])$, we have

    $$p_\omega(C'[P] + R, S_P, O) = p_\omega(R, S_P, O) = p_\omega(C'[Q] + R, S_P, O)$$

  Let us consider now the (**must**) case. Like in the (**may**) case, we have two possibilities, corresponding to the applications of SUM1 and SUM2 respectively. With SUM2 the proof is exactly the same. With SUM1, we proceed in an analogous way and we derive that, for every scheduler S_Q of $T(C'[Q])$ there exists a scheduler S_P of $T(C'[P])$ such that

  $$p_\omega(C'[P], S_P, O) \leq p_\omega(C'[Q], S_Q, O) = p_\omega(C'[Q] + R, S_Q, O)$$

  Note now that the process $T(C'[P])$ cannot be blocked, because $C' = a.C''$ for some $a \neq \tau$, and $T(C'[Q])$ is not blocked. Hence S_P is a non-blocking scheduler for $T(C[P])$, and we have

  $$p_\omega(C'[P] + R, S_P, O) = p_\omega(C'[P], S_P, O)$$

Note that the restriction to guarded + is essential. Otherwise $T(C'[P])$ can be blocked, thus S_P will be blocking for $T(C[P])$ so we will be forced to use a scheduler for $T(C[P])$ that chooses R.

- Case $C = (\nu a)C'$
  The process $(\nu)((\nu a)C'[X] \mid O)$ has the same transitions as $(\nu)(C'[X] \mid (\nu a)O)$. The result follows from the induction hypothesis.

We have shown the proposition for any context where $[\,]$ is not under replication. Now we show it for an arbitrary context C by induction on the number k of nested bangs that enclose $[\,]$. The base case $k = 0$ has already been shown. Consider a context $C = !l{:}a.C'$ with k nested bangs. First, we define:

$$C[P]^m = \begin{cases} l{:}a.(\rho_0 C'[P] \mid \rho_1 C[P]^{m-1}) & m > 1 \\ l{:}a.\rho_0 C'[P] & m = 1 \end{cases}$$

Intuitively, $C[P]^m$ is the m-times unfolding of $C[P] = !l : a.C'[P]$, taking into account the relabeling that takes place each time a new process is spawned. Then we prove that:

$$C[P]^m \sqsubseteq_{\mathbf{may}} C[Q]^m \qquad \forall m \geq 1 \tag{12}$$

The proof is by induction on m (this is nested, part of the proof of the inductive case for the induction on k). It is easy to see that

$$P \sqsubseteq_{\mathbf{may}} Q \quad \Rightarrow \quad \rho_i P \sqsubseteq_{\mathbf{may}} \rho_i Q \qquad i \in \{0, 1\} \tag{13}$$

For the base case $m = 1$ we have $C'[P] \sqsubseteq_{\mathbf{may}} C'[Q]$ (hypothesis of the outer induction) thus $\rho_0 C'[P] \sqsubseteq_{\mathbf{may}} \rho_0 C'[Q]$ from (13) and finally $l{:}a.\rho_0 C'[P] \sqsubseteq_{\mathbf{may}} l{:}a.\rho_0 C'[Q]$ (apply context $l{:}a.[\,]$).

For the inductive case we have

$$\begin{aligned} C[P]^{m-1} \sqsubseteq_{\mathbf{may}} C[Q]^{m-1} \quad &\Rightarrow && \text{Ind. Hyp.} \\ \rho_1 C[P]^{m-1} \sqsubseteq_{\mathbf{may}} \rho_1 C[Q]^{m-1} \quad &\Rightarrow && (13) \\ \rho_0 C'[P] \mid \rho_1 C[P]^{m-1} \sqsubseteq_{\mathbf{may}} \rho_0 C'[P] \mid \rho_1 C[Q]^{m-1} && && \text{Cont. without !} \end{aligned}$$

The last step is obtained by applying the context $\rho_0 C'[P] \mid [\,]$. Note that this context has no bangs enclosing $[\,]$, and its labeling is fresh wrt $\rho_1 C[P]^{m-1}$ and $\rho_1 C[Q]^{m-1}$. Then we have

$$\begin{aligned} \rho_0 C'[P] \sqsubseteq_{\mathbf{may}} \rho_0 C'[Q] \quad &\Rightarrow && \text{Outer Ind. Hyp., (13)} \\ \rho_0 C'[P] \mid \rho_1 C[Q]^{m-1} \sqsubseteq_{\mathbf{may}} \rho_0 C'[Q] \mid \rho_1 C[Q]^{m-1} && && \text{Cont. without !} \end{aligned}$$

Finally by the transitivity of $\sqsubseteq_{\mathbf{may}}$ and by applying the context $l{:}a.[\,]$ we get

$$l{:}a.(\rho_0 C'[P] \mid \rho_1 C[P]^{m-1}) \sqsubseteq_{\mathbf{may}} l{:}a.(\rho_0 C'[Q] \mid \rho_1 C[Q]^{m-1})$$

thus $C[P]^m \sqsubseteq_{\mathbf{may}} C[Q]^m$. This concludes the proof of (12) (inner induction). Finally, assuming the negation of our claim, we have $C[P] \not\sqsubseteq_{\mathbf{may}} C[Q]$, that is

$$\exists O \exists S_P \; \forall S_Q \; : \; p_\omega(C[P], S_P, O) > p_\omega(C[Q], S_Q, O)$$

There can be executions containing ω of arbitrary length, however their probability will go to zero as the length increases. Thus there will be an m such that if we consider only executions of length at most m then the above inequality will still hold. But these executions can be also performed by $C[P]^m, C[Q]^m$ which contradicts (12). This concludes the outer induction.

Similarly for $\sqsubseteq_{\mathbf{must}}$ in the case of replicated input. □

This also implies that $\approx_{\mathbf{may}}, \approx_{\mathbf{must}}$ are congruences. Note that P, Q in the above proposition are not required to have linear labelings, P might include multiple occurrences of the same label thus limiting the power of the schedulers S_P. This shows the locality of the scheduler's restriction: some choices inside P are hidden from the scheduler but the rest of the context is fully visible.

If we remove the freshness condition of the context then Proposition 7 is no longer true in the presence of probabilistic choice. Let $P = l_1 : a.l_2 : b$, $Q = l_3 : a.l_4 : b$ and $C = l : (l_1 : a.l_2 : c +_p [\,])$. We have $P \approx_{\mathbf{may}} Q$ but $C[P], C[Q]$ can be separated by the test $O = \bar{a}.\bar{b}.\omega \mid \bar{a}.\bar{c}.\omega$ (when the labeling is omitted assume a linear one). It is easy to see that $C[Q]$ can pass the test with probability 1 by selecting the correct branch of O based on the outcome of the probabilistic choice. In $C[P]$ this is not possible because of the labels l_1, l_2 that are common in P, C.

Note also that the restriction to guarded-sum contexts in the case of (**must**) is essential. As a counterexample, consider the processes $P = 0$ and $Q = \tau.0$. We have that $P \sqsubseteq_{\mathbf{must}} Q$. However, if $C = [\,] + a.0$, then $C[P] \not\sqsubseteq_{\mathbf{must}} C[Q]$, the witness being $O = \bar{a}.\omega$.

We can now state the result that we announced in Section 1.

Theorem 8. *Let P, Q be CCS_σ processes and C a context with a linear and fresh labeling (wrt P, Q, l, l_1) and without occurrences of bang. Then*

$$\begin{array}{lll} l{:}(C[l_1{:}\tau.P] +_p C[l_1{:}\tau.Q]) & \approx_{\mathit{may}} & C[l{:}(P +_p Q)] \quad \textit{and} \\ l{:}(C[l_1{:}\tau.P] +_p C[l_1{:}\tau.Q]) & \approx_{\mathit{must}} & C[l{:}(P +_p Q)] \end{array}$$

If $+_p$ does not occur in C then the restriction on C's labeling can be dropped.

Proof. Since we will always use the label l for each probabilistic sum $+_p$, and l_1 for $\tau.P$ and $\tau.Q$, we will omit these labels to make the proof more readable. We will also denote $(1-p)$ by $\bar{p}$.

Let $R_1 = C[\tau.P] +_p C[\tau.Q]$ and $R_2 = C[P +_p Q]$. We will prove that for all tests O and for all schedulers $S_1 \in Syn((\nu)(R_1 \mid O))$ there exists $S_2 \in Syn((\nu)(R_2 \mid O))$ such that $p_\omega(R_1, S_1, O) = p_\omega(R_2, S_2, O)$ and vice versa. This implies both $R_1 \approx_{\mathbf{may}} R_2$ and $R_1 \approx_{\mathbf{must}} R_2$.

Without loss of generality we assume that tests do not perform internal actions, but only synchronizations with the tested process. First, it is easy to see that

$$p_\omega(P +_p Q, l.S, O) = p\ p_\omega(P, S, O) + \bar{p}\ p_\omega(Q, S, O) \quad (14)$$

$$p_\omega(l_1{:}a.P, (l_1, l_2).S, O) = p_\omega(P, S, O') \quad (15)$$

where $(\nu)(l_1{:}a.P \mid O) \parallel (l_1, l_2).S \xrightarrow{\tau} \delta((\nu)(P \mid O')) \parallel S$.

In order for the scheduler of R_1 to be non-blocking, it has to be of the form $l.S_1$, since the only possible transition of R_1 is the probabilistic choice labeled by l. By (14) we have

$$p_\omega(C[\tau.P] +_p C[\tau.Q], l.S_1, O) = p\ p_\omega(C[\tau.P], S_1, O) + \bar{p}\ p_\omega(C[\tau.Q], S_1, O)$$

The proof will be by induction on the structure of C. Let O range over tests, let S_1 range over non-blocking schedulers for both $C[\tau.P]$ and $C[\tau.Q]$ (such that $l.S_1$ is a non-blocking scheduler for R_1) and let S_2 range over non-blocking schedulers for R_2. The induction hypothesis is:

$\Rightarrow$) $\forall O\ \forall S_1\ \exists S_2$:
$p\ p_\omega(C[\tau.P], S_1, O) + \bar{p}\ p_\omega(C[\tau.Q], S_1, O) = p_\omega(C[P +_p Q], S_2, O)$ and
$\Leftarrow$) $\forall O\ \forall S_2\ \exists S_1$:
$p\ p_\omega(C[\tau.P], S_1, O) + \bar{p}\ p_\omega(C[\tau.Q], S_1, O) = p_\omega(C[P +_p Q], S_2, O)$

We have the following cases for C (note that we use the restriction on the labeling only for the probabilistic choice):

- Case $C = [\,]$. Trivial.
- Case $C = l_1{:}a.C'$
  The scheduler S_1 of $C[\tau.P]$ and $C[\tau.Q]$ has to be of the form $S_1 = (l_1, l_2).S_1'$ where l_2 is the label of a $\overline{a}$ prefix in O (if no such prefix exists then the case is trivial).

  A scheduler of the form $(l_1, l_2).S$ can schedule any process of the form $l_1{:}a.X$ (with label l_1) giving the transition:

  $$(\nu)(l_1{:}a.X \mid O) \parallel (l_1, l_2).S \xrightarrow{\tau} \delta((\nu)(X \mid O')) \parallel S$$

  and producing always the same O'. The probability p_ω for these processes will be given by equation (15).

  Thus for ($\Rightarrow$) we have

  $$\begin{aligned}
  & p\ p_\omega(l_1{:}a.C'[\tau.P], (l_1, l_2).S_1', O) + \bar{p}\ p_\omega(l_1{:}a.C'[\tau.Q], (l_1, l_2).S_1', O) \\
  &= p\ p_\omega(C'[\tau.P], S_1', O') + \bar{p}\ p_\omega(C'[\tau.Q], S_1', O') && (15) \\
  &= p_\omega(C'[P +_p Q], S_2', O') && \text{Ind. Hyp.} \\
  &= p_\omega(l_1{:}a.C'[P +_p Q], (l_1, l_2).S_2', O) && (15) \\
  &= p_\omega(R_2, S_2, O)
  \end{aligned}$$

For ($\Leftarrow$) we can perform the above derivation in the opposite direction, given that a scheduler for $R_2 = l_1 : a.C'[P +_p Q]$ must be of the form $S_2 = (l_1, l_2).S'_2$.

- Case $C = C' \mid R$
  We have that $R \mid O$ is itself a test, and

$$p_\omega(X \mid R, S, O) = p_\omega(X, S, R \mid O) \tag{16}$$

  Thus for ($\Rightarrow$) we have

$$\begin{array}{lll} & p\ p_\omega(C'[\tau.P] \mid R, S_1, O) + \bar{p}\ p_\omega(C'[\tau.Q] \mid R, S_1, O) & \\ = & p\ p_\omega(C'[\tau.P], S_1, R \mid O) + \bar{p}\ p_\omega(C'[\tau.Q], S_1, R \mid O) & (16) \\ = & p_\omega(C'[P +_p Q], S_2, R \mid O) & \text{Ind. Hyp.} \\ = & p_\omega(C'[P +_p Q] \mid R, S_2, O) & (16) \\ = & p_\omega(R_2, S_2, O) & \end{array}$$

  For ($\Leftarrow$) we can perform the above derivation in the opposite direction.

- Case $C = l_1{:}(C' +_q R)$
  Since we consider only contexts with linear and fresh labelings, the labels of $C'[X]$ are disjoint from those of R, thus the scheduler of a process of the form $l_1{:}(C'[X] +_q R)$ must be of the form $S = l_1.(\textbf{if } l_C \textbf{ then } S_C \textbf{ else } S_R)$ where $l_C \in tl(C'[X])$, S_C is a scheduler containing labels of $C'[X]$ and S_R is a scheduler containing labels of R. Moreover

$$\begin{array}{lll} & p_\omega(l_1{:}(C'[X] +_q R), S, O) & \\ = & q\ p_\omega(C'[X], \textbf{if } l_C \textbf{ then } S_C \textbf{ else } S_R, O) + & \\ & \bar{q}\ p_\omega(R, \textbf{if } l_C \textbf{ then } S_C \textbf{ else } S_R, O) & \\ = & q\ p_\omega(C'[X], S_C, O) + \bar{q}\ p_\omega(R, S_R, O) & (17) \end{array}$$

  As a consequence, the scheduler S_1 of $C[\tau.P]$ and $C[\tau.Q]$ has to be of the form $S_1 = l_1.(\textbf{if } l_C \textbf{ then } S_C \textbf{ else } S_R)$. Note that $tl(C'[\tau.P]) = tl(C'[\tau.Q])$ so the two processes cannot be separated by a test. S_C will schedule both (possibly separating them later).

  For ($\Rightarrow$) we have

$$\begin{array}{lll} & p\ p_\omega(l_1{:}(C'[\tau.P] +_q R), S_1, O) + \bar{p}\ p_\omega(l_1{:}(C'[\tau.Q] +_q R), S_1, O) & \\ = & q(p\ p_\omega(C'[\tau.P], S_C, O) + \bar{p}\ p_\omega(C'[\tau.Q], S_C, O)) + & \\ & \bar{q}\ p_\omega(R, S_R, O) & (17) \\ = & q\ p_\omega(C'[P +_p Q]), S'_C, O) + & \\ & \bar{q}\ p_\omega(R, S_R, O) & \text{Ind. Hyp.} \\ = & p_\omega(l_1{:}(C'[P +_p Q] +_q R), l_1.(\textbf{if } l'_C \textbf{ then } S'_C \textbf{ else } S_R), O) & (17) \\ = & p_\omega(R_2, S_2, O) & \end{array}$$

  Where $l'_C \in tl(C'[P +_p Q])$ (and thus $l'_C \notin tl(R)$).

For ($\Leftarrow$) we can perform the above derivation in the opposite direction, given that a scheduler for $R_2 = l_1{:}(C'[P +_p Q] +_q R)$ must be of the form $S_2 = l_1.(\textbf{if } l'_C \textbf{ then } S'_C \textbf{ else } S_R)$.

- Case $C = C' + R$
  Consider the process $C'[l_0{:}\tau.P] + R$. The scheduler S_1 of this process has to choose between $C'[l_0{:}\tau.P]$ and R.

  There are two cases to have a transition using the SUM1, SUM2 rules.

  a) Either $S_1 = S_R$ and

  $$\text{SUM2}\ \frac{(\nu)(R \mid O) \parallel S_R \xrightarrow{\alpha} \mu \parallel S'_R}{(\nu)(C'[l_0{:}\tau.P] + R \mid O) \parallel S_R \xrightarrow{\alpha} \mu \parallel S'_R}$$

  In this case

  $$p_\omega(C'[l_0{:}\tau.P] + R, S_R, O) = p_\omega(R, S_R, O) \tag{18}$$

  b) Or $S_1 = S_C$ and

  $$\text{SUM1}\ \frac{(\nu)(C'[l_0{:}\tau.P] \mid O) \parallel S_C \xrightarrow{\alpha} \mu \parallel S'_C}{(\nu)(C'[l_0{:}\tau.P] + R \mid O) \parallel S_C \xrightarrow{\alpha} \mu \parallel S'_C}$$

  In this case

  $$p_\omega(C'[l_0{:}\tau.P] + R, S_C, O) = p_\omega(C'[l_0{:}\tau.P], S_C, O) \tag{19}$$

  Now consider the process $C'[l_0{:}\tau.Q] + R$. Since P and Q are behind the $l_0{:}\tau$ action, we have $tl(C'[l_0{:}\tau.Q] = tl(C'[l_0{:}\tau.P])$. Thus S_R and S_C will select R and $C'[l_0{:}\tau.Q]$ respectively and the equations (18) and (19) will hold.

  In the case (a) ($S = S_R$) we have:

  $$\begin{aligned}
  & p\ p_\omega(C'[\tau.P] + R, S_R, O) + \bar{p}\ p_\omega(C'[\tau.Q] + R, S_R, O) \\
  = \ & p\ p_\omega(R, S_R, O) + \bar{p}\ p_\omega(R, S_R, O) \quad (18) \\
  = \ & p_\omega(R, S_R, O) \\
  = \ & p_\omega(C'[P +_p Q] + R, S_R, O) \\
  = \ & p_\omega(R_2, S_2, O)
  \end{aligned}$$

  In the case (b) ($S = S_C$) we have:

  $$\begin{aligned}
  & p\ p_\omega(C'[\tau.P] + R, S_C, O) + \bar{p}\ p_\omega(C'[\tau.Q] + R, S_C, O) \\
  = \ & p\ p_\omega(C'[\tau.P], S_C, O) + \bar{p}\ p_\omega(C'[\tau.Q], S_C, O) \quad (19) \\
  = \ & p_\omega(C'[P +_p Q], S'_C, O) \qquad \text{Ind. Hyp.} \\
  = \ & p_\omega(C'[P +_p Q] + R, S'_C, O) \\
  = \ & p_\omega(R_2, S_2, O)
  \end{aligned}$$

  For ($\Leftarrow$) we can perform the above derivation in the opposite direction.

- Case $C = (\nu a)C'$
  The process $(\nu)((\nu a)C'[X] \mid O)$ has the same transitions as $(\nu)(C'[X] \mid (\nu a)O)$. The result follows from the induction hypothesis.

□

There are two crucial points in the above theorem. The first is that the labels of the context are copied, thus the scheduler cannot distinguish between $C[l_1{:}\tau.P]$ and $C[l_1{:}\tau.Q]$ based on the labels of the context. The second is that P, Q are protected by a τ action labeled by the same label l_1. This is to ensure that in the case of a nondeterministic sum ($C = R + [\,]$) the scheduler cannot find out whether the second operand of the choice is P or Q unless it commits to selecting the second operand. For example, let $R = a +_{0.5} 0$, $P = a$, $Q = 0$ (all omitted labels are linear). Then $R_1 = (R + P) +_{0.1} (R + Q)$ is not testing equivalent to $R_2 = R + (P +_{0.1} Q)$ since they can be separated by $O = \overline{a}.\omega$ and a scheduler that resolves $R + P$ to P and $R + Q$ to R (it will be of the form **if** l_P **then** S_P **else** S_R). However, if we take $R'_1 = (R + l_1{:}\tau.P) +_{0.1} (R + l_1{:}\tau.Q)$ then R'_1 is testing equivalent to R_2 since now the scheduler cannot see the labels of P, Q so if it selects P then it is bound to also select Q.

The problem with replication is simply the persistence of the processes. Clearly $(!a.P) +_p (!a.Q)$ cannot be equivalent to $!a.(P +_p Q)$, since the first replicates only one of P, Q while the second replicates both. However Theorem 8 together with Proposition 7 imply that

$$C'[l{:}(C[l_1{:}\tau.P] +_p C[l_1{:}\tau.Q])] \approx_{\mathbf{may}} C'[C[l{:}(P +_p Q)]] \tag{20}$$

where C is a context without bang and C' is an arbitrary context. The same is also true for $\approx_{\mathbf{must}}$ (under the extra condition that $+$ occurs only guarded in C'). This means that we can lift the sum towards the root of the context until we reach a bang. Intuitively we cannot move the sum outside the bang since each replicated copy must perform a different probabilistic choice with a possibly different outcome.

Theorem 8 shows that the probabilistic choice is indeed private to the process and invisible to the scheduler. The process can perform it at any time, even in the very beginning of the execution, without making any difference to an outside observer.

6. An application to security

In this section we discuss an application of our framework to anonymity. In particular, we show how to specify the Dining Cryptographers protocol ([24]) so that it is robust to scheduler-based attacks. We first propose a method to encode *secret value passing*, which will turn out to be useful for the specification.

6.1. Encoding secret value passing

We propose to encode the passing of a secret message as follows:

$$\begin{aligned} l{:}c(x).P &\triangleq \textstyle\sum_{v\in V} l{:}c_v.P[v/x] \\ l{:}\bar{c}\langle v\rangle.P &\triangleq l{:}\overline{c_v}.P \end{aligned}$$

where V is the finite set of values that can be transmitted through channel c. This is the usual encoding of value passing in CCS: we use a nondeterministic sum with a distinct channel c_v for each v. The novelty is that we use the same label in all the branches of the nondeterministic sum. To ensure that the resulting labeling is deterministic we should restrict the channels c_v and make sure that there is at most one output on c. We will write $(\nu c)P$ for $(\nu_{v\in V} c_v)P$. For example, the labeling of the following process is deterministic:

$$(\nu c)(l_1{:}c(x).P \mid l{:}(l_2{:}\bar{c}\langle v\rangle +_p l_2{:}\bar{c}\langle w\rangle))$$

This case is a combination of the cases (5) and (7) of Proposition 5. The two outputs on c are on different branches of the probabilistic sum, so during an execution at most one of them will be available. Thus there is no ambiguity in scheduling the sum produced by $c(x)$. The scheduler $l.(l_1, l_2)$ will perform a synchronization on c_v or c_w, whatever is available after the probabilistic choice. In other words, using the labels we manage to hide the information about which value was transmitted to P.

6.2. Dining cryptographers with a probabilistic master

The problem of the Dining Cryptographers is the following: Three cryptographers dine together. After the dinner, the bill has to be paid either by one of them or by another agent called the master. The master decides who will pay and then informs each cryptographer separately about whether they have to pay or not. The cryptographers would like to find out whether the payer is the master or one of them. However, in the latter case, they also wish to keep the payer anonymous.

The Dining Cryptographers Protocol (DCP) solves the above problem as follows: each cryptographer tosses a fair coin which is visible to himself and his neighbor to the right. Each cryptographer checks the two adjacent coins and, if he is not paying, announces *agree* if they are the same and *disagree* otherwise. However, the paying cryptographer says the opposite. It can be proved that the master is paying if and only if the number of *disagrees* is even ([24]).

An external observer O is supposed to see only the three announcements $\overline{out_i}\langle\ldots\rangle$. As discussed in [25], DCP satisfies strong anonymity. However, this analysis considers only the value that each cryptographer announces, without considering the order in which they make their announcements. In other words, the announcement *aad* is considered to be the same, whether it corresponds to $c_1 = a, c_2 = a, c_3 = d$ or to $c_2 = a, c_3 = d, c_1 = a$ (in the indicated order).

If we want to allow the cryptographers to make announcements in any order, then the only reasonable way to model the choice of order is nondeterministically. But this leads immediately to a simple attack: if the scheduler

$$
\begin{aligned}
Master &\triangleq l_1 : \textstyle\sum_{i=0}^{2} p_i(\underbrace{\overline{m}_0\langle i == 0\rangle}_{l_2} \mid \underbrace{\overline{m}_1\langle i == 1\rangle}_{l_3} \mid \underbrace{\overline{m}_2\langle i == 2\rangle}_{l_4}) \\
Crypt_i &\triangleq \underbrace{m_i(pay)}_{l_{5,i}} . \underbrace{c_{i,i}(coin_1)}_{l_{6,i}} . \underbrace{c_{i,i\oplus 1}(coin_2)}_{l_{7,i}} . \underbrace{\overline{out}_i\langle pay \otimes coin_1 \otimes coin_2\rangle}_{l_{8,i}} \\
Coin_i &\triangleq l_{9,i} : ((\underbrace{\bar{c}_{i,i}\langle 0\rangle}_{l_{10,i}} \mid \underbrace{\bar{c}_{i\ominus 1,i}\langle 0\rangle}_{l_{11,i}}) +_{0.5} (\underbrace{\bar{c}_{i,i}\langle 1\rangle}_{l_{10,i}} \mid \underbrace{\bar{c}_{i\ominus 1,i}\langle 1\rangle}_{l_{11,i}})) \\
Prot &\triangleq (\nu \vec{m})(Master \mid (\nu \vec{c})(\textstyle\prod_{i=0}^{2} Crypt_i \mid \prod_{i=0}^{2} Coin_i))
\end{aligned}
$$

Figure 5: Encoding of the dining cryptographers with probabilistic master

is unrestricted then it can base its strategy on the decision of the master, by selecting the paying cryptographer last (or first). Clearly, an external observer would trivially identify the payer just from the fact that he spoke last. A similar situation would arise if the scheduler based its decision on the value of the coins.

A natural question to ask at this point is whether this attack is realistic, or just an artifact of the nondeterministic model. For instance, is it possible for the scheduler to know the decision of the master? The answer is that this attack could appear in practice without even a malicious intention from the part of the scheduler. For example, the payer needs to make one more calculation to add 1 to his message, so he might need more time to make his announcement and, as a consequence, he will be scheduled last by a simple "first comes first served" scheduler.

In any case, the scheduler restrictions, if any, should be part of the requirements when stating the anonymity properties of a protocol. For example the analysis should state "assuming that the coins are fair and that the scheduler's decisions are independent from the master's choice and from the coins, DCP satisfies strong anonymity". This way an implementor of the protocol will have to verify that the scheduler condition is satisfied, or somehow assume that it is.

In our framework we can solve the problem by giving a specification of the DCP in which the choices of the master and of the coins are made invisible to the scheduler. The specification is shown in Figure 5. We use some meta-syntax for brevity: The symbols $\oplus$ and $\ominus$ represent the addition and subtraction modulo 3, while $\otimes$ represents the addition modulo 2 (xor). The notation $i == n$ stands for 1 if $i = n$ and 0 otherwise.

There are many sources of nondeterminism: the order of communication between the master and the cryptographers, the order of reception of the coins, and the order of the announcements. The crucial points of our specification, which make the nondeterministic choices independent from the probabilistic ones, are: (a) all communications internal to the protocol (master-cryptographers and cryptographers-coins) are done by secret value passing, and (b) in each probabilistic choice the different branches have the same labels. For example, all

branches of the master contain an output on m_0, always labeled by l_2, but with different values each time.

Thanks to the above independence, the specification satisfies strong anonymity. There are various equivalent definitions of this property. We follow here the version presented in [25]. Let $\vec{o}$ represent an observable (the sequence of announcements), and $p_S(\vec{o} \mid \overline{m_i}\langle 1\rangle)$ represent the conditional probability, under the scheduler S, that the protocol produces $\vec{o}$ given that the master has selected Cryptographer i as the payer.

Proposition 9 (Strong anonymity). *The protocol in Figure 5 satisfies the following property: for all schedulers S and for all observables $\vec{o}$:*

$$p_S(\vec{o} \mid \overline{m}_0\langle 1\rangle) = p_S(\vec{o} \mid \overline{m}_1\langle 1\rangle) = p_S(\vec{o} \mid \overline{m}_2\langle 1\rangle)$$

Proof. Since the process is finite, it contains a finite number of nondeterministic choices and as a result the set of its schedulers is also finite. Thus, the proposition can be verified by calculating the probability of all traces under all schedulers (this could be even done automatically). Here we make a higher level argument to show that the proposition holds.

Let v_1, v_2, v_3 be the values announced by the cryptographers, that is, v_i is the output of the subprocess $\overline{out}_i\langle pay \otimes coin_1 \otimes coin_2\rangle$. These values depend only on the selection of the master (pay) and the outcome of the coins ($coin_1, coin_2$) and not on the scheduler, the latter can only affect their order. From the proof of strong anonymity for a fixed announcement order ([24]) we know that $p(v_1, v_2, v_3|a_i) = p(v_1, v_2, v_3|a_j)$ for all cryptographers i, j and all values of v_1, v_2, v_3.

Now the observables of the protocol are of the form $\vec{o} = \overline{out_{k_1}}\langle v_{k_i}\rangle, \overline{out_{k_2}}\langle v_{k_2}\rangle, \overline{out_{k_3}}\langle v_{k_3}\rangle$ where k_1, k_2, k_3 is the index of the cryptographer who speaks first, second and third, respectively. The order (that is, the k_i's) depends on the scheduler. However, in all random choices the same labels appear in both branches of the choice, so a scheduler cannot use an **if-then-else** test to "detect" the outcome of the choice (it would be useless since the same branch of the **if** would be always activated). As a consequence, the order is fixed for a particular scheduler, that is, a scheduler uniquely defines the k_i's above. With a fixed order, the probability of each $\vec{o}$ is equal to the probability of the corresponding v_i's, thus

$$p_S(\vec{o} \mid \overline{m}_i\langle 1\rangle) = p(v_1, v_2, v_3|a_i) = p(v_1, v_2, v_3|a_j) = p_S(\vec{o} \mid \overline{m}_j\langle 1\rangle)$$

□

Note that different schedulers will produce different traces (we still have nondeterminism) but they will not depend on the choice of the master.

Some previous treatment of the DCP, including [25], had solved the problem of the leak of information due to too-powerful schedulers by simply considering sets of announcements as observable, rather than sequences of announcements. Thus, one could think that using a true concurrent semantics, for instance event

$$\begin{array}{lll} P & ::= \ldots \mid l{:}\{P\} & \\ CP & ::= P \parallel S, T & \end{array} \qquad \text{SWITCH} \quad \frac{P \parallel T, 0 \xrightarrow{\tau} \mu \parallel T', 0}{l{:}\{P\} \parallel l.S, T \xrightarrow{\tau} \mu \parallel S, T'}$$

Figure 6: Adding an "independent" scheduler to the calculus

structures, would solve the problem. There are two issues with this approach: first, by abstracting away from the interleaving we weaken the model too much. There is an important information, namely the order in which the messages are sent on the network, that is present in reality but not in a true concurrent model, possibly leading to missed attacks. Second, the problem of the scheduler arises from any form of nondeterminism, not only from the interleaving. Thus, in cases like the anonymity example of the introduction, abstracting from the interleaving will not solve the problem.

6.3. Dining cryptographers with a nondeterministic master

Up to now we considered the master in the dining cryptographers to be probabilistic, that is, we assume that the master makes his decision using some probability distribution. An interesting question is whether we can remove this assumption, that is, make the same analysis with a nondeterministic master. However, this case poses a conceptual problem: as we discussed in the previous paragraph, the decision of the master should be invisible to the scheduler. But if the master is nondeterministic then the scheduler itself will make the decision, so how is it possible for a scheduler to be oblivious to his own choices?

We sketch here a method to hide also certain nondeterministic choices from the scheduler. First we need to extend the calculus with the concept of a second *independent* scheduler T that we assume to resolve the nondeterministic choices that we want to make transparent to the main scheduler S. The new syntax and semantics are shown in Figure 6. The construct $l : \{P\}$ represents a process where the scheduling of P is protected from the main scheduler S. The scheduler S can "ask" T to schedule P by selecting the label l. Then the schedulers switch roles and T resolves the nondeterminism of P acting as the main scheduler, as expressed by the SWITCH rule. Note that we need to add T to all the other rules of the semantics, in all these rules T is simply inactive. Moreover, we assume that T does not collaborate with S, so we can use labels in P freely without revealing information to the scheduler.

To model the dining cryptographers with nondeterministic master we replace the $Master$ process in Figure 5 by the following one.

$$Master \triangleq l_1{:}\Big\{\textstyle\sum_{i=0}^{2} l_{12,i}{:}\tau.(\underbrace{\overline{m}_0\langle i == 0\rangle}_{l_2} \mid \underbrace{\overline{m}_1\langle i == 1\rangle}_{l_3} \mid \underbrace{\overline{m}_2\langle i == 2\rangle}_{l_4})\Big\}$$

Essentially we have replaced the probabilistic choice by a *protected* nondeterministic one. Note that the labels of the operands are different but this is not a problem since this choice will be scheduled by T. Note also that after the choice

we still have the same labels l_2, l_3, l_4. However the labeling is still deterministic, similarly to the case (6) of Proposition 5.

In case of a nondeterministic selection of the anonymous events, and a probabilistic anonymity protocol, the notion of strong anonymity has not been established yet, although some possible definitions have been discussed in [25]. Our framework makes it possible to give a natural and precise definition.

In the probabilistic case of the previous section, we compared the conditional probabilities $p_S(\vec{o} \mid \overline{m}_0\langle 1\rangle)$ and $p_S(\vec{o} \mid \overline{m}_1\langle 1\rangle)$ corresponding to different choices of the master. Now the choice of cryptographer i is made by the secondary scheduler $T_i = l_{12,i}$, so instead of conditional probabilities, we will have probabilities of the form $p_{S,T_i}(\vec{o})$, where $p_{S,T_i}(\cdot)$ is the probability measure on traces induced by the schedulers S, T_i. Then we naturally arrive at the following definition.

Definition 6 (Strong anonymity for nondeterministic anon. events). A protocol with a nondeterministic selection of the anonymous event satisfies strong anonymity iff for all observables $\vec{o}$, schedulers S, and independent schedulers T_i, T_j (selecting different anonymous events), we have:

$$p_{S,T_i}(\vec{o}) = p_{S,T_j}(\vec{o})$$

We can now show that the above property holds for our protocol:

Proposition 10. *The DCP with nondeterministic master, specified in this section, satisfies strong anonymity.*

Proof. Similar to Proposition 9, since $p_{S,T_i}(\vec{o})$ is equal to $p_S(\vec{o} \mid \overline{m}_i\langle 1\rangle)$ in the protocol with a probabilistic master. □

7. Conclusion and Future work

We have proposed a process-calculus approach to the problem of limiting the power of the scheduler so that it does not reveal the outcome of hidden random choices, and we have shown its applications to the specification of information-hiding protocols. We have also discussed a feature, namely the distributivity of certain contexts over random choices, that makes our calculus appealing for verification. Finally, we have considered the probabilistic testing preorders and shown that they are precongruences in our calculus.

Our plans for future work are in various directions: first, we would like to investigate the possibility of giving a game-theoretic characterization of our notion of scheduler. A first step in this direction has been made in [26]. Second, we want to study the use of equivalences to define security properties, while coping at the same time with the problem of the scheduler. This direction has been explored in [27]. Finally, we would like to investigate whether methods to restrict the scheduler, like those presented in this paper, can be used in a probabilistic model checker like PRISM. Currently, PRISM considers all possible schedulers when verifying a formula, without the possibility to restrict to a subset of them.

Acknowledgments

We wish to thank Vincent Danos for having pointed out to us an attack to the Dining Cryptographers protocol based on the order of the scheduler, which has inspired this work. We also thank Roberto Segala and Daniele Varacca for their valuable comments on a previous version of this paper. Finally, we wish to express our gratitude to the anonymous reviewers, for their thorough work and their recommendations, that have helped to improve the paper considerably.

[1] M. Y. Vardi, Automatic verification of probabilistic concurrent finite-state programs, in: Proceedings of the 26th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Portland, Oregon, 1985, pp. 327–338.

[2] H. Hansson, B. Jonsson, A framework for reasoning about time and reliability, in: Proceedings of the 10th IEEE Symposium on Real-Time Systems, IEEE Computer Society Press, Santa Monica, California, USA, 1989, pp. 102–111.

[3] W. Yi, K. G. Larsen, Testing probabilistic and nondeterministic processes, in: Proceedings of the 12th IFIP International Symposium on Protocol Specification, Testing and Verification, North Holland, Florida, USA, 1992, pp. 47–61.

[4] R. Segala, Modeling and verification of randomized distributed real-time systems, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, available as Technical Report MIT/LCS/TR-676 (Jun. 1995).

[5] R. Segala, N. Lynch, Probabilistic simulations for probabilistic processes, Nordic Journal of Computing 2 (2) (1995) 250–273.

[6] H. Hansson, B. Jonsson, A calculus for communicating systems with time and probabilities, in: Proceedings of the Real-Time Systems Symposium - 1990, IEEE Computer Society Press, Lake Buena Vista, Florida, USA, 1990, pp. 278–287.

[7] E. Bandini, R. Segala, Axiomatizations for probabilistic bisimulation, in: Proceedings of the 28th International Colloquium on Automata, Languages and Programming, Vol. 2076 of Lecture Notes in Computer Science, Springer, 2001, pp. 370–381.

[8] S. Andova, Probabilistic process algebra, Ph.D. thesis, Technische Universiteit Eindhoven (2002).

[9] M. Mislove, J. Ouaknine, J. Worrell, Axioms for probability and nondeterminism, in: F. Corradini, U. Nestmann (Eds.), Proc. of the 10th Int. Wksh. on Expressiveness in Concurrency (EXPRESS '03), Vol. 96 of Electronic Notes in Theoretical Computer Science, Elsevier, 2004, pp. 7–28.

[10] C. Palamidessi, O. M. Herescu, A randomized encoding of the π-calculus with mixed choice, Theoretical Computer Science 335 (2-3) (2005) 373–404.

[11] Y. Deng, C. Palamidessi, J. Pang, Compositional reasoning for probabilistic finite-state behaviors, in: A. Middeldorp, V. van Oostrom, F. van Raamsdonk, R. C. de Vrijer (Eds.), Processes, Terms and Cycles: Steps on the Road to Infinity, Vol. 3838 of Lecture Notes in Computer Science, Springer, 2005, pp. 309–337.

[12] A. Sokolova, E. d. Vink, Probabilistic automata: system types, parallel composition and comparison, in: C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, M. Siegle (Eds.), Validation of Stochastic Systems: A Guide to Current Research, Vol. 2925 of Lecture Notes in Computer Science, Springer, 2004, pp. 1–43.

[13] B. Jonsson, K. G. Larsen, W. Yi, Probabilistic extensions of process algebras, in: J. A. Bergstra, A. Ponse, S. A. Smolka (Eds.), Handbook of Process Algebra, Elsevier, 2001, Ch. 11, pp. 685–710.

[14] K. Chatzikokolakis, Probabilistic and information-theoretic approaches to anonymity, Ph.D. thesis, LIX, École Polytechnique (Oct. 2007).

[15] J. Aranda, C. D. Giusto, C. Palamidessi, F. Valencia, Expressiveness of recursion, replication and scope mechanisms in process calculi, in: F. S. de Boer, M. M. Bonsangue (Eds.), Postproceedings of the 5th International Symposium on Formal Methods for Components and Objects (FMCO'06), Vol. 4709 of Lecture Notes in Computer Science, Springer, 2007, pp. 185–206.

[16] F. D. Garcia, P. van Rossum, A. Sokolova, Probabilistic anonymity and admissible schedulers, arXiv:0706.1019v1 (2007).

[17] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, R. Segala, Task-structured probabilistic I/O automata, in: Proceedings the 8th International Workshop on Discrete Event Systems (WODES'06), Ann Arbor, Michigan, 2006.

[18] R. Canetti, L. Cheung, D. K. Kaynar, M. Liskov, N. A. Lynch, O. Pereira, R. Segala, Time-bounded task-PIOAs: A framework for analyzing security protocols, in: S. Dolev (Ed.), Proceedings of the 20th International Symposium in Distributed Computing (DISC '06), Vol. 4167 of Lecture Notes in Computer Science, Springer, 2006, pp. 238–253.

[19] L. de Alfaro, T. A. Henzinger, R. Jhala, Compositional methods for probabilistic systems, in: K. G. Larsen, M. Nielsen (Eds.), Proceedings of the 12th International Conference on Concurrency Theory (CONCUR 2001), Vol. 2154 of Lecture Notes in Computer Science, Springer, 2001, pp. 351–365.

[20] G. Boudol, I. Castellani, A non-interleaving semantics for CCS based on proved transitions, Fundamenta Informaticae XI (1988) 433–452.

[21] C. Bodei, P. Degano, C. Priami, Mobile processes with a distributed environment, in: F. M. auf der Heide, B. Monien (Eds.), Proceedings of the 23rd International Colloquium on Automata, Languages and Programming (ICALP'96), Vol. 1099 of LNCS, Springer-Verlag, Berlin-Heidelberg-New York-London-Paris-Tokyo-Hong Kong-Barcelona-Budapest-Milan-Santa Clara-Singapore, 1996, pp. 490–501.

[22] R. D. Nicola, M. C. B. Hennessy, Testing equivalences for processes, Theoretical Computer Science 34 (1-2) (1984) 83–133.

[23] M. Abadi, A. D. Gordon, A calculus for cryptographic protocols: The spi calculus, Information and Computation 148 (1) (1999) 1–70.

[24] D. Chaum, The dining cryptographers problem: Unconditional sender and recipient untraceability, Journal of Cryptology 1 (1988) 65–75.

[25] M. Bhargava, C. Palamidessi, Probabilistic anonymity, in: M. Abadi, L. de Alfaro (Eds.), Proceedings of CONCUR, Vol. 3653 of Lecture Notes in Computer Science, Springer, 2005, pp. 171–185.

[26] K. Chatzikokolakis, S. Knight, P. Panangaden, Epistemic strategies and games on concurrent processes, in: M. Nielsen, A. Kucera, P. B. Miltersen, C. Palamidessi, P. Tuma, F. D. Valencia (Eds.), SOFSEM, Vol. 5404 of Lecture Notes in Computer Science, Springer, 2009, pp. 153–166.

[27] K. Chatzikokolakis, G. Norman, D. Parker, Bisimulation for demonic schedulers, in: L. de Alfaro (Ed.), FOSSACS, Vol. 5504 of Lecture Notes in Computer Science, Springer, 2009, pp. 318–332.